PROGRAMADORES

Número 49
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Coordinador Técnico
Eduardo de Riquer
eriquer@towercom.es
Redactora Jefe
Cristina Peña
crisp@towercom.es

Colaboradores
Constantino Sánchez Ballesteros, Enrique de la Lastra, Ignacio Rojo, Enrique Díaz, Jordi Agost, Enrique de Alarcón, Jorge Delgado, Javier Sanz, Francisco M. Rosado.

Maquetación
Javier Pérez Manzanaro
Tratamiento de Imagen
María Arce Giménez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29 - 970 47 44 35

Suscripciones
Begoña Álvarez
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Óscar Rodríguez (jefe)
oscarrf@towercom.es
Javier Amado
jamado@towercom.es
Servicio Técnico
Óscar Casado
ocasado@towercom.es

Preimpresión
Indes Color
Impresión
Gráficas Muriel
Distribución
SGEL
Distribución en Argentina / Chile / Colombia / México / Venezuela
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

La revista Sólo Programadores no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados. El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 31-12-98

Editorial

Planificación digital

Puede que llegue un día (seguro que no muy lejano) en que algunas conversaciones alcancen un trasfondo entre gracioso, esperpéntico y lamentable, similares a las que a continuación les comento:

- Así que me dijo su nombre, dirección, estado civil y número de hijos, pero ¿cuántos ordenadores tiene en casa?.

- Pues con los que tengo en el trastero dispongo de seis.

- Hombre, pues para no estar casado es usted un poco viciosillo. Y dígame, ¿a cuál quiere usted más?.

- La verdad es que para mí son todos iguales, aunque últimamente hago más caso al pequeño, de edad, que no de prestaciones, porque me ha salido muy listo. De vez en cuando me dice que le duele un error de protección general, aunque un médico muy inteligente con gafas me ha asegurado que con Windows 98 se le pasará. Los mayores dan menos guerra, sobre todo desde que se hicieron amigos de un tal Linux, que aunque es un poco serio, los deja seguir trabajando como antes.

A nadie con un mínimo de experiencia en este mundo le puede sorprender una respuesta así, sobre todo recordando los equipos que almacena en su casa, con sistemas operativos de toda índole y con una red infinita que se extiende desde el balcón al cuarto de baño. Pero la situación no se detiene en este punto, sino que se agrava ante la posibilidad de aprovechar la ropa (perdón, los componentes) de los mayores para que se puedan utilizar con los jovenzuelos de la familia. Es tal el grado de desesperación, que un amigo le comentaba a otro:

- Ya no aguanto más, no lo soporto, qué hago con el Pentium del mediano, no hay manera de colocárselo al Slot 1 de la placa del recién nacido.

- No te preocupes, yo tengo un primo que seguro que puede hacerlo.

- ¿Seguro? ¿Qué pasa, es ingeniero industrial o algo así?

- Que va, pero lleva un montón de años en un gimnasio, y seguro que no se le resiste.

Unos días después recordaban con lágrimas (de alegría o desesperación, a elegir) el chispazo en el momento del implante y comentaban: me has matado a un ordenador, pero lo que nos hemos reido...

Resulta demencial, pero ha llegado el momento de realizar un control de natalidad de tipo digital, para no incurrir en situaciones tan ridículas como las descritas. Seguro que si les digo que están basadas en la realidad, más de uno podría sentirse traicionado por algún confidente deslenguado. Tendremos que tener cuidado con lo que contamos por ahí y sobre todo, acercarnos a las tiendas de informática sin olvidar el preservativo.

solo PROGRAMADORES

49

6 Noticias
LO ULTIMO EN DESARROLLO

Las noticias más actuales llegan a esta sección para que nuestro lector esté al día sobre todo aquello que está ocurriendo en el campo que más le interesa: las novedades de la tecnología informática y todo su entorno; herramientas, empresas, actualidad, etc.

8 Java
CONCURRENCIA Y SINCRONIZACIÓN DE THREADS (II)

Si en el anterior número tomamos un primer contacto con lo que son los threads y sus posibilidades, ahora nos centramos en cómo éstos son capaces de compartir datos y los diferentes modos que existen de hacerlo.

16 Linux
MÁS ALLÁ DE SVGALIB: VGAGL (III)

Una vez conocidos los elementos básicos de la libre de Linux SVGALIB, iremos un poco más lejos en e tercera entrega para completar nuevos conocimient Descubriremos VGAGL, y todas las posibilidades ésta nos ofrece a la hora de crear aplicaciones gráfi

25 C/C++
PROGRAMACIÓN BÁSICA EN WINDOWS (III)

Siguiendo con el curso de C bajo Windows, y tras plicar las nociones más elementales veremos m fondo todas las líneas de código de un ejemplo d programa básico C para Windows.

37 Herramientas
COMPARATIVA DE RDA JAVA

Cinco herramientas elementales para todos aquellos que trabajan con el lenguaje Java: Visual Age 1.0, Visual J++ 6.0, Visual Café 2.5, J Builder 2 y Java Studio 1.0. Todas ellas quedan expuestas en estas páginas, sus posibilidades, carencias, aplicaciones y secretos. Paso a paso vamos a ir examinando y contrastando las ventajas e inconvenientes de cada una de ellas.

30
Visual Basic
PROGRAMACIÓN DEL API DE WINDOWS (II)

Profundizaremos en las llamadas a la API y así adquirir una sólida base teórica para después ser capaces de realizar ejemplos prácticos de nuestros programas. Gracias a este artículo aprenderemos cómo pasar matrices a un procedimiento de una DLL o crear guías de los errores más usuales en las llamadas a la API.

56
Programación multimedia
RECONOCIMIENTO DE VOZ (II)

Tras haber visto las nociones más básicas dentro de este campo, nos vamos a adentrar en cuestiones tan importantes como las características de los micrófonos y las tarjetas de sonido, o los errores que pueden llegar a cometer los engines. También vamos a acercarnos a los comandos usados para la programación en Visual Basic de la API Speech.

62
C++ Builder 3,
CREACIÓN DE COMPONENTES (II)

Si en la entrega anterior nuestra misión fue abordar de forma genérica el tema de los componentes, en este segundo capítulo nos adentraremos en la creación manual de uno de ellos mediante un asistente, de forma que paso a paso logremos nuestro objetivo.

68
Redes locales
CABLEADO DE RED

Un nuevo artículo de gran interés para nuestros lectores: el cableado de las redes locales. Conoceremos diversos aspectos relacionados con este tema a través de la descripción de las tipologías y sus distintas adecuaciones en los diferentes casos.

74
Libros
NUEVAS PUBLICACIONES

"Sólo Programadores" busca todos los meses aquellos libros que os puedan interesar respecto a los temas de plena actualidad. En esta ocasión manuales para mejorar las páginas web, para ampliar los conocimientos en el desarrollo de 3D, para conocer a fondo Internet Explorer 4 o para descubrir Visual InterDev.

47
Programación de sockets
INTEGRACIÓN DE CGI EN EL SERVIDOR WEB (y V)

Damos por finalizada esta interesante serie que nos ha llevado a profundizar en el conocimiento de un servidor web. Todo aquello que tenía relación con él ha quedado aclarado: su capacidad de diálogo, las respuestas a las peticiones realizadas por los clientes. Así hemos logrado dotarle de una actividad que antes no poseía.

76
CORREO DEL LECTOR

En esta sección se pretende dar respuesta a todas las dudas que puedan tener nuestros lectores, pudiendo dirigirse a los propios autores o bien a la siguiente dirección de correo de nuestra publicación: solop@towercom.es.

78
CONTENIDO DEL CD-ROM

Y el complemeto perfecto para nuestra revista se haya en el interior del CD-ROM que le acompaña, ahora con un interface actualizado. En este número hallaréis utilidades tan interesantes como el Oracle Proxy Server 1.0., Apache Web Server 1.3.1. o Microsoft DirectX 6.0 SDK. Ahora sólo queda que lo descubráis vosotros mismos.

Permite a los fabricantes integrar soluciones para aprovechar los sitemas de hardware existentes

SUN ANUNCIA LA DISPONIBILIDAD DE JAVAPC

Sun Microsystems anunció la disponibilidad de JavaPC Engine 1.0, una solución software que convierte en network computers a los PCs Intel 486 y Pentium con DOS y Windows 3.x.

Este software permite a los fabricantes de equipos originales (OEMs) integrar soluciones para aprovechar los sistemas de hardware existentes y estandarizar sus aplicaciones en toda la red, con la consiguiente reducción de los costes asociados a la formación, soporte y administración.

Con JavaPC Engine 1.0, los usuarios pueden utilizar en cualquier punto de la red aplicaciones compatibles con el software Java Development Kit (JDK 1.1), a la vez que conservan el acceso a programas en DOS y Windows. Su configuración flexible permite que los responsables de los sistemas de información decidan el ritmo de transición desde la informática basada en PCs a los más asequibles entornos de networking computing.

Se trata de un software de fácil instalación, que tan sólo necesita PCs estándar y no sobreescribe los archivos ni las aplicaciones actuales de los usuarios que lo utilizan.

Además de todo esto también ofrece un entorno básico, que incluye en su estructura un sencillo módulo de arranque de aplicaciones, que facilita que los administradores de sistemas configuren el entorno de red o las aplicaciones verticales que hayan decidido.

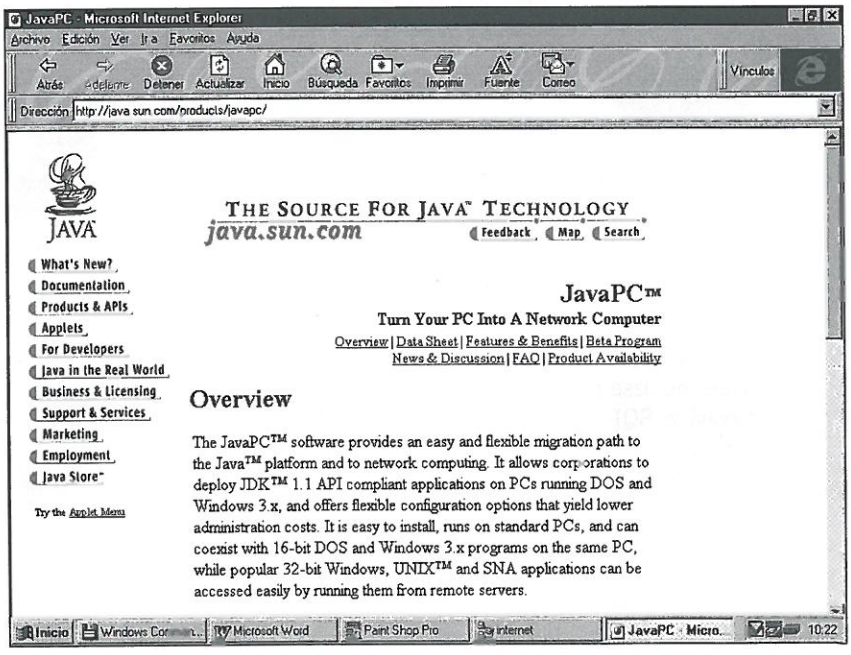
Incluye en su estructura bibliotecas de clases y la máquina virtual Java, lo que permite a los usuarios ejecutar aplicaciones basadas en el software JDK 1.1.

Su funcionamiento está basado en el mismo sistema operativo que potencia los network computers utiliza controladores DOS para hacer de interfaz con los periféricos de los PCs. Con ello, se garantiza al usuario un alto nivel de compatibilidad con la mayoría de los PCs que ejecutan DOS, incluídos los que ejecutan Windows 3.x.

Este producto se ha consolidado hoy por hoy como uno de los entornos de desarrollo más novedoso creado por la empresa Sun.

Esta beta permite la creación de máquinas virtuales que logran la ejecución de cualquier programa en lenguaje Java.

Si está interesado en este producto, puede obtener más información en la siguiente página: http://www.java.-sun.com/ products/javapc.


JavaPC - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Imprimir Fuente Correo
Vínculos
Dirección http://java.sun.com/products/javapc/
JAVA
THE SOURCE FOR JAVA™ TECHNOLOGY
java.sun.com
Feedback Map Search
What's New?
Documentation
Products & APIs
Applets
For Developers
Java in the Real World
Business & Licensing
Support & Services
Marketing
Employment
Java Store™
Try the Applet Menu
JavaPC™
Turn Your PC Into A Network Computer
Overview | Data Sheet | Features & Benefits | Beta Program
News & Discussion | FAQ | Product Availability
Overview
The JavaPC™ software provides an easy and flexible migration path to the Java™ platform and to network computing. It allows corporations to deploy JDK™ 1.1 API compliant applications on PCs running DOS and Windows 3.x, and offers flexible configuration options that yield lower administration costs. It is easy to install, runs on standard PCs, and can coexist with 16-bit DOS and Windows 3.x programs on the same PC, while popular 32-bit Windows, UNIX™ and SNA applications can be accessed easily by running them from remote servers.
Inicio
Windows Co...
Microsoft Word
Paint Shop Pro
Internet
JavaPC - Micro...
10:22

TIVOLI LANZA UNA NUEVA VERSIÓN DE NETVIEW PARA OS/390

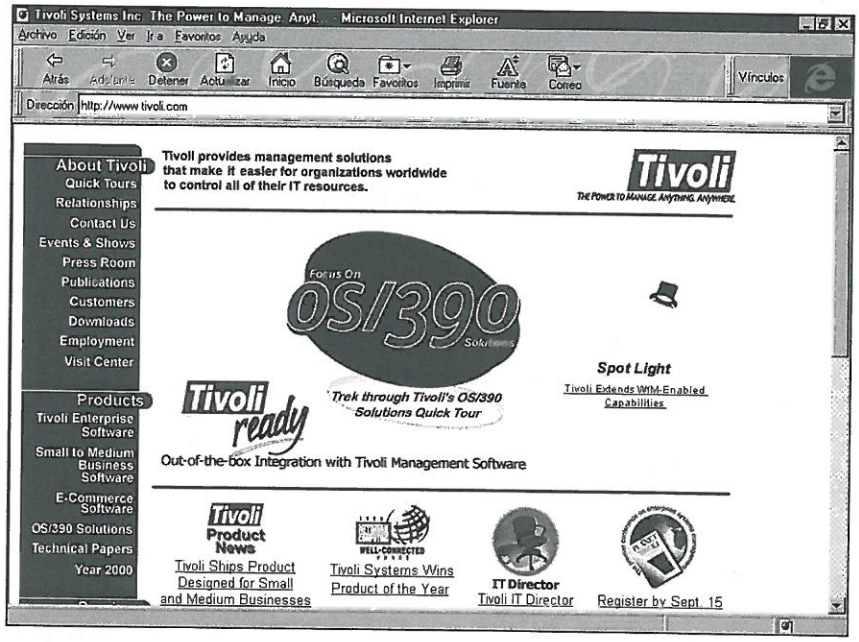
Proporciona una gestión total de sistemas mainframe en entornos distribuidos

Tivoli Systems, líder mundial en software de gestión de sistemas, redes y aplicaciones, anuncia la disponibilidad en el mercado de una nueva versión de Tivoli NetView para OS/390, UNIX y NT.

Esta última versión permite a los clientes la gestión centralizada de la red corporativa desde el mainframe, incluyendo los protocolos TCP/IP, IPX y SNA, con las ventajas añadidas de disponer de una consola de gestión que puede estar basada en Java o en un navegador.

Gracias al uso de nuevas herramientas de validación, Tivoli NetView para OS/390 reducirá drásticamente el tiempo dedicado a desarrollar y desplegar complejas rutinas de automatización.

Además de todas estas ventajas permite la utilización de una sola plataforma para gestionar mejor su red desde sus sistemas basados en OS/390, a la misma vez que logra reducir el tiempo dedicado a resolver problemas que puedan surgir en la gestión de los sistemas y la red.



PERVASIVE SOFTWARE AMPLÍA SU SOPORTE PARA WINDOWS

Pervasive Software ha anunciado que el motor de base de datos ultra ligero Pervasive. SQL ha pasado con éxito las pruebas de compatibilidad para la plataforma Windows 98. Las exhaustivas pruebas incluyeron el funcionamiento de la actual línea de productos de Pervasive en servidores NT y NetWare, así como versiones anteriores de los productos de bases de datos de Pervasive. La arquitectura Pervasive.SQL permite a las aplicaciones cliente/servidor empaquetadas acceder simultáneamente a un motor de base de datos único con una capacidad de proceso transaccional de alto volumen y muy rápido.

Las principales características que incluye esta última versión 1.2 son por ejemplo la consola de gestión de Tivoli basada en Java, la cual está diseñada como una alternativa a la NGMF existente, además de contar con un interfaz de usuario gráfico muy intuitivo.

También cabe destacar la automatización total de entorno de test, donde los clientes pueden evaluar complicadas rutinas de trabajo de forma totalmente automatizada antes de su aplicación real ,o el acceso basado en navegador entre otras cosas.

Puede ampliar toda esta información en **www.tivoli.com/**

Concurrencia y sincronización de threads (II)

Javier Sanz Alamillo (jsanz@a01-unix.uc3m.es)

A lo largo de este artículo vamos a tratar sobre cómo los threads pueden compartir datos de una forma correcta, sin problemas, utilizando los diferentes métodos disponibles para ello. Además se detallarán los modos para la comunicación e intercambio de datos entre éstos.

La utilización de *threads* que comparten datos puede dar a lugar a situaciones anómalas cuya solución exige el uso de un mecanismo de sincronización de acceso al recurso compartido. El lenguaje Java ofrece unos sencillos mecanismos para solucionar este problema, por lo que la solución suele resultar por regla general bastante simple y efectiva.

SINCRONIZACIÓN Y COMUNICACIÓN ENTRE THREADS

Estos mecanismos se basan en el concepto de "monitores", un esquema ampliamente utilizado y que fue desarrollado por *C.A.R. Hoare*. No es necesario tener una comprensión exacta de este mecanismo, pero es de gran ayuda para conseguir un entendimiento de los problemas que se abordarán.

Un monitor es esencialmente un cerrojo, que se aplica a un determinado recurso que es compartido, de forma que a este recurso pueden acceder múltiples *threads*, pero únicamente por uno de ellos a la vez.

El mecanismo de cerrojos se puede entender con un ejemplo de la realidad. Supongamos que tenemos un hotel con una única habitación libre. Cuando la ocupe un inquilino, a este inquilino se le da una llave que le permite disfrutar de esa habitación, pero en cuanto se marcha y devuelve la llave, esa habitación esta disponible para otros posibles inquilinos, de manera que la posesión de la llave determina quién esta usando el recurso, o sea, la habitación.

El problema clásico que determina el uso de cerrojos es el siguiente. Supongamos que dos *thread* se alternan en el incremento del contenido, de una variable común, un contador. En un determinado momento, el *thread* A lee el contenido del contador, lo incrementa en uno y va a guardar en la variable el nuevo contenido. Se puede dar el caso de que en ese momento el sistema pase al *thread* de activo a dormido (se le acabó su tiempo), y no ha podido guardar el valor correcto del contador.

Ahora le toca el turno al *thread* B, que verá el mismo contenido que el *thread* anterior, debido a que éste no le dio tiempo a guardar el resultado correcto. A continuación *thread* el actual incrementará el contador y guardará su resultado. Ahora se le acaba el

tiempo, el sistema ejecuta el *thread* A que en vez de incrementar el valor según lo dejó el *thread* B escribirá su valor calculado anteriormente y continuará donde terminó su ejecución, escribiendo su propio resultado, por lo que tenemos que los dos *thread* en vez de incrementar el contenido de una variable en dos unidades lo han hecho en uno.

El uso de recursos compartidos usando threads puede dar lugar a situaciones anómalas

Afortunadamente Java se encarga de todo el proceso de sincronización, ejecuta internamente todo el mecanismo de acceso, manipula el estado del cerrojo, o sea, su bloqueo y correcto desbloqueo, por lo que las aplicaciones sólo tienen que indicar los recursos que se van a bloquear.

LA PALABRA CLAVE SYNCHRONIZED

Por regla general se suele necesitar sincronización entre *threads* cuando queremos acceder ordenadamente a un recurso compartido, normalmente un objeto. En otras palabras, la sincronización se encarga de asegurar que un *thread* en un determinado momento pueda manipular un objeto. En Java, cada objeto tiene un cerrojo asociado con él. Para ser más específicos, cada clase y cada instancia de la clase tiene su propio cerrojo. Gracias a ello, se puede utilizar la palabra clave *synchronized* que se utiliza para definir qué parte de código (en realidad un método completo) puede ser accedido por un único *thread*, por lo que sé encarga de gestionar el mecanismo de cerrojos.

UN EJEMPLO: EL TOTAL BANCARIO

Desarrollemos un ejemplo que ponga en práctica la teoría. Supongamos un banco que consta de dos cajeros que se encargan de contar billetes de un valor cualquiera en pesetas. Cada uno de ellos tiene que contar una cantidad de billetes y cuando ha finalizado su tarea muestra el número total de billetes que ha contado y la suma correspondiente. Si hacemos que los dos cajeros sean dos *threads* y que cuenten el número de billetes y den la suma de los que han contado, podremos poner en práctica la sincronización.

En el primer caso vamos a hacer que cada cajero cuente su propio montón de billetes, para mostrar los cambios y cómo es posible el cambio de los resultados:

```
import java.io.*;
import java.lang.*;

class Banco {
      public static void main ( String args[]) {
       try {
              // Declaramos los dos montones
      de billetes
              Contador co1 = new Contador
      ();
              Contador co2 = new Contador
      ();
              // Declaramos los dos cajeros
              Cajero c1 = new Cajero(co1);
              Cajero c2 = new Cajero(co2);

              // Se ponen a contar..
              c1.start();
              c2.start();
              c1.join();
              c2.join();
       }
       catch ( Exception e ){
              e.printStackTrace();
       }
      }
}
```

La clase *Contador* se encargará de contar los billetes y almacenar la suma y el total de billetes contados. En este caso concreto se supondrá que el montón es de billetes de 2.000 pesetas.

```
class Contador {
      int numBilletes = 0 ;
      long suma = 0 ;
      final int TOTAL_BILLETES = 10000 ;
      final int VALOR_BILLETES = 2000 ;
      void cuenta () {
       // A contar la suma de los billetes
       for ( numBilletes =0 ; numBilletes <
      TOTAL_BILLETES ; numBilletes ++ )
      {
              suma += VALOR_BILLETES ;
      // Billetes de 2000 pts
              Thread.yield();
       }
       System.out.println ( numBilletes+ "
      suman : "+ suma + " pts");
      }
}
```

En la siguiente clase declaramos los *thread* que serán los cajeros que cuentan. Cada cajero recibe un montón para contar, o sea, un objeto de tipo *Contador*:

```
class Cajero extends Thread {

      Contador contadorCajero ;
      Cajero ( Contador paramContador ) {
       contadorCajero = paramContador ;
      }

      public void run () {
       contadorCajero.cuenta();
      }
}
```

Se obtiene lo siguiente:

```
10000 suman: 20000000 pts
10000 suman: 20000000 pts
```

Es un resultado correcto, ya que cada *thread* tenía un montón de billetes diferentes para contar.

Ahora supongamos que los dos cajeros deben contar del mismo montón, o sea, lo comparten y por tanto, la suma de lo que haya contado cada uno debe ser el resultado total.

Para ello, modificaremos el código añadiendo lo siguiente:

```
// Declaramos los dos montones de billetes
Contador co1 = new Contador ();
// Ahora sobra, Contador co2 = new Contador ();
// Declaramos los dos cajeros y el mismo montón.
Cajero c1 = new Cajero(co1);
Cajero c2 = new Cajero(co1);
```

Con este cambio obtenemos:

```
10000 suman: 20002000 pts
10001 suman: 20002000 pts
```

Observando el resultado, comprobamos que los dos cajeros no saben contar y además, un cajero ha contado un billete más que el otro, y encima el total sumado es el mismo en ambos casos, por lo que está claro que ha producido un error de sincronismo. En un momento un *thread* ha interrumpido a otro en una parte crítica, en la suma o en la cuenta de los billetes.

Un cerrojo es un mecanismo que aplicado sobre un recurso permite un acceso correcto a éste por parte de los threads

Por tanto, debemos utilizar un mecanismo de sincronización que garantice que cuando un cajero cuente un billete y lo sume, el otro no pueda intentar coger el mismo billete y sumarlo. La solución que ofrece Java para resolver este problema es de lo más simple y eficiente, utilizando la cláusula *synchronized* en la declaración del método donde se realiza la tarea "crítica".

Por tanto, cambiaremos el método *void cuenta()* por:

```
synchronized void cuenta ()
```

Si realizamos ese cambio, obtenemos el siguiente resultado:

```
10000 suman : 20000000 pts
10000 suman : 40000000 pts
```

Es el resultado esperado. Una vez que el primer cajero entra, no acaba hasta que termina de contar, debido a que mientras no finalice su trabajo no ha salido del método, impidiendo que el otro cajero pudiera tocar el montón de billetes para contar, porque hemos declarado el método *cuenta()* con la cláusula *synchronized.*

De todos modos, hay un detalle que se tiene que explicar. ¿Por qué el cajero que finaliza cuenta 40000000 en vez de 20000000?. Esto ocurre porque no se inicializa la variable *suma* antes del bucle que cuenta los billetes, por lo que el segundo cajero continúa la suma en donde la dejó el anterior.

Si modificamos el código e incluimos la inicialización, tendremos:

```
void cuenta () {
// Cada cajero cuenta lo suyo ...
    suma = 0 ;
    // A contar la suma de los billetes
    for ( numBilletes =0 ; numBilletes <
    TOTAL_BILLETES ; numBilletes ++ )
    {
        suma += VALOR_BILLETES ;

Thread.yield();
    }
```

A partir de este momento obtenemos el siguiente resultado esperado tal y como detallamos:

```
10000 suman : 20000000 pts
10000 suman : 20000000 pts
```

Otra forma de realizar la sincronización consiste en declarar el objeto compartido como sincronizado en vez del método que lo contiene. Se realiza entonces el siguiente cambio:

```
public void run(){
    contadorCajero.cuenta();
}
```

por:

```
public void run(){
  synchronized (contadorCajero ) {
        contadorCajero.cuenta();
  }
}
```

Se ejecuta esta nueva modificación y se obtiene el mismo resultado. En este caso lo que se está sincronizando es el acceso al objeto, no el método que lo contiene.

El uso de cerrojos permite la comunicación entre threads de forma que uno puede bloquear a otro debido a diferentes causas, como una espera, un cálculo, etc

La cuestión está en que si un método es de tipo *synchronized* y si se ejecuta sobre un objeto en un *thread*, otro *thread* no puede llamar a este método sobre ese objeto, pero puede llamar a otro método sobre otro objeto.

Recordando el caso en que el método estaba declarado con *synchronized,* esto nos garantizaba que dos *thread* no podían ejecutar a la vez ninguna parte del método, pero sobre el mismo objeto, no sobre diferentes objetos. Si declaramos dos objetos diferentes podríamos tener los mismos problemas.

Además, la cláusula *synchronized* permite ejecutar dos métodos diferen-

tes sobre el mismo objeto o el mismo método sobre dos objetos diferentes, pero impide que más de un *thread* ejecute la misma porción de código al mismo tiempo sobre el mismo objeto.

La cláusula synchronized permite sincronizar el acceso a un determinado método

Llegados a este punto debemos de destacar que el hecho de usar *synchronized* de una forma poco práctica puede conducirnos a interbloqueos y provoca una degradación del rendimiento del código, cerca de cuatro veces, por lo que es preferible utilizar esta palabra clave sólo para métodos cortos y en aquellos que se consideran como imprescindibles.

UN POCO MÁS ALLÁ. LOS MÉTODOS WAIT() Y NOTIFY()

Si bien mediante la utilización de la cláusula *synchronized* podemos tener un acceso sincronizado a un método o un objeto de un bloque de código, el uso de *wait()* y *notify()* permite ampliar esta posibilidad. Estos métodos pertenecen de por sí a la clase *Object*. Cada objeto en Java es una subclase de *Object*, por lo que gracias a la herencia podemos utilizar sus métodos. Usando wait() y *notify()* un *thread* puede detenerse sobre un cerrojo en un punto cualquiera que podemos definir, y entonces puede esperar a que otro *thread* le notifique que puede continuar. Todo esto ocurre dentro de bloques sincronizados y se cumple que un único *thread* puede ejecutarse a la vez. La ventaja de este método es que ofrece mayor flexibilidad de bloqueo y una forma de controlar los *thread* de manera más práctica.

La ejecución del método *wait()* en un bloque sincronizado determinado permite que el *thread* quede en un modo de "espera o bloqueado" sobre un determinado cerrojo y por tanto pasa al estado de "dormido". Un *thread* puede necesitar ser bloqueado porque tiene que esperar a que otro *thread* haya terminado de realizar una tarea en otra parte del programa, o se cumpla una determinada condición, etc. Entonces, cuando se cumpla dicha condición o la tarea que espera se haya realizado, el *thread* que estaba en "espera" por el bloqueo es "desbloqueado" mediante una llamada al método *notify()* por el *thread* que estaba activo, y el que estaba dormido pasará a estado de activo, tomando el control sobre el cerrojo correspondiente.

Cuando el *thread* que quedó bloqueado toma el control sobre el cerrojo prosigue con la siguiente sentencia donde se quedó, esto es, la siguiente sentencia tras *wait*(). De todos modos, hay que tener en cuenta que esto depende de cuando el *thread* que libera el cerrojo por el uso del método *notity*() ya que si no se realiza la llamada el *thread* bloqueado no continuaría ejecutándose. Por ello, existe una versión sobrecargada del método *wait*() que permite especificar un determinado tiempo de espera, a partir del cual el *thread* que está bloqueado despierta. Tiene la ventaja de que se evitan bloqueos extraños pero no garantiza un uso correcto de los cerrojos.

Un thread demonio es un hilo que ofrece servicios a otros

Vamos a desarrollar un ejemplo sencillo en el que se utilicen *wait* y *notify* para poder apreciar la utilidad de estos métodos. Supongamos que tenemos unos alumnos de una clase esperando a que el profesor les comunique sus notas. Los dos *threads* de *Alumnos* y el *thread Profesor* compartirán el objeto *Nota*:

```
import java.io.* ;

class escuela {
      public static void main ( String args[] ){
       try {
            // Generamos el objeto Nota, el
      profesor
            // y los dos alumnos ...
            Nota laNota = new Nota ();
            Profesor p = new Profesor (
      laNota );
            Alumno a = new Alumno (
      "Javi", laNota);
            Alumno b = new Alumno (
      "Jose", laNota );
            // Empezamos la ejecución
            a.start();
            b.start();
            p.start();
            a.join();
            b.join();
            p.join();
       }
       catch ( Exception e ){
            System.out.println ( e.toString()
      );
       }
      }
}
class Alumno extends Thread{
      Nota na ; // nota del alumno
      String nom ; // nombre

      Alumno ( String nombre , Nota n )
      {
       na = n ;
       nom = nombre ;
      }

      public void run (){
       System.out.println ( nom + " Esperado
      su nota" );
       na.esperar(); // el alumno espera la nota
       System.out.println ( nom + " recibio su
      nota");  }
}
```

```
class Profesor extends Thread{
        Nota na ;

        Profesor ( Nota n ){
         na = n ;
        }
        public void run (){
         System.out.println ( " Voy a poner la
        nota ");
        na.dar (); // el profesor pone la nota del
        alumno
        }

}

class Nota {
        synchronized void esperar ()      {
         try {
                  wait();
         }
         catch (InterruptedException e ){
                  ;
         }
        }
        synchronized void dar (){
         notifyAll();
        }
}
```

El resultado que se obtiene es el siguiente:

```
Javi Esperado su nota
Jose Esperado su nota
Voy a poner la nota
Javi recibio su nota
Jose recibio su nota
```

Expliquemos la manera en que se desarrolla el programa. Una vez que se han generado los *threads* para los alumnos y el profesor, los alumnos ejecutan el método *na.esperar()* por lo que llaman al método *wait*() y quedan esperando a que el profesor ponga la nota y llame al método *notify()*. Esto lo realiza en el método *na.dar()*.

Obsérvese que no se ha ejecutado *notify()* sino *notifyAll()*. Si se hubiera ejecutado el primero se habría desbloqueado a un *thread*, a un único alumno, y el otro quedaría esperando. Puesto que lo que hay que hacer es desbloquear más de un *thread* se utiliza el método *notifyAll()*.

Los *threads Alumnos*, una vez que han sido desbloqueados por el profesor ordenadamente, reciben su nota, es decir, continúan su ejecución en la línea siguiente de código.

EL EJEMPLO CLÁSICO DEL CONSUMIDOR Y PRODUCTOR

El ejemplo clásico en el tema de la comunicación entre dos *threads* es el del consumidor y el productor. Un *thread* productor genera un mensaje y lo pone en una cola. Mientras otro *thread,* el consumidor, recoge ese mensaje y lo visualiza. Tenemos un productor que tiene una cola limitada para generar mensajes y para desarrollar un ejemplo claro, haremos que el consumidor sea más lento que el productor, por lo que éste a veces tendrá que parar hasta que el consumidor acabe con algunos mensajes de la cola y se lo comunique .

Por tanto, tendremos un *thread* productor que genera una serie de mensajes, parando si llena una cola. Cada vez que genera un mensaje se lo notifica al consumidor y continúa con su trabajo, generando más mensajes.

Por el otro lado, el consumidor se encarga de recibir los mensajes y mostrarlos, por lo que dormirá mientras no tenga nada que recibir. Ahora bien, ¿qué ocurre cuando el productor esta durmiendo debido a que el consumidor es muy lento?. Pasemos a ver el código de ejemplo:

```
import java.util.Vector;
import java.util.Date;
class productor extends Thread {
        static final int MAXELEM=1;
        private Vector mensajes = new Vector();

        public void run() {
         try {
            while ( true ) {
             crearMensaje(); //genera mensaje
             sleep ( 1000 ); // a dormir
            }
         }
         catch ( InterruptedException e ) { }

        }

        private synchronized void crearMensaje()
        throws InterruptedException
        {
         // Mientras que la cola este llena, a
        esperar ...
         while ( mensajes.size() == MAXELEM
        )
                  wait();

         // genero el mensaje, lo añado a la cola
         mensajes.addElement ( new Date().toS-
        tring() );

         // Aviso al consumidor de que hay un
        nuevo mensaje
         notify();
        }

        // Atencion, este método es llamado
        SOLO por el CONSUMIDOR
        public synchronized String recibirMensa-
        je() throws InterruptedException
        {

         // Mientras no haya nada que recoger ...
        dormir ...
         while ( mensajes.size() == 0 )
                  wait();

         // Eliminamos el mensaje leido y lo
        devolvemos
         String cadMensaje = (String) mensa-
        jes.firstElement();
         mensajes.removeElement( cadMensaje
        );

         // Aviso al productor que se ponga a tra-
        bajar si se durmió ya que puede añadir un
```

```
        elemento
         notify();

         return cadMensaje;
        }
}

class consumidor extends Thread {
        productor elProductor ;

        consumidor ( productor unProductor ) {
         elProductor = unProductor ;
        }

        public void run () {
         try {
            while ( true ) {
                String cadMensaje= elProduc-
        tor.recibirMensaje();
                System.out.println ( " Recibido
        del productor " + cadMensaje );
                sleep( 2000 ); // Consumidor a
        dormir ...
            }
         }
         catch ( InterruptedException e ) { }

        }

        public static void main ( String args[] )
        {
         productor miProductor = new produc-
        tor();
         miProductor.start();
         new consumidor( miProductor ).start();
        }
}
```

Ejecutamos el código y obtenemos lo siguiente:

```
Recibido del productor Sat Jun 27 18:34:51
      GMT+02:00 1998
Recibido del productor Sat Jun 27 18:34:52
      GMT+02:00 1998
Recibido del productor Sat Jun 27 18:34:53
      GMT+02:00 1998
Recibido del productor Sat Jun 27 18:34:55
      GMT+02:00 1998
...
```

Analicemos el código. El código del productor consta de una cola de mensajes y se dedica a crear mensajes y almacenarlos. Tras esto se pone a dormir. El método *crearMensaje()* se encarga de la creación de los mismos, comprobando si ya la cola de mensajes ya está llena, ya que si no tendría que bloquearse al no poder crear más mensajes. Este bloqueo estaría activo hasta que el consumidor avisara al productor de que no tiene nada que recoger. Además, el método almacena el mensaje y comunica el consumidor que ya tiene la posibilidad de recoger un mensaje nuevo.

El método *recibirMensaje()* que es invocado por el consumidor, nunca por el productor, se encarga de la tarea contraria. En caso de que no haya ningún mensaje, el consumidor espera, por lo que se bloquea llamando a *wait()*. Una vez que hay un mensaje y el productor lo ha desbloqueado, lo recoge y lo elimina de la cola del productor. A su vez, siempre que elimina un elemento notifica al productor que despierte por si está durmiendo y así éste se pone a generar nuevos mensajes. Finaliza visualizándolo y haciendo una pausa, ya que hemos hecho que el productor sea más veloz que el consumidor.

Vamos a añadir un segundo consumidor al programa, para observar qué ocurre. Cambiamos la clase consumidor de forma que contenga el siguiente código y modificamos el método *crearMensaje* del productor para que su notificación sea una llamada al método *notifyAll()*:

```
private synchronized void crearMensaje()
        throws InterruptedException
        {
         // Mientras que la cola este llena, a
        esperar ...
         while ( mensajes.size() == MAXELEM
        )
                wait();

         // genero el mensaje, lo añado a la cola
         mensajes.addElement ( new Date().toS-
        tring() );

         // Aviso a los consumidores de que hay
          un nuevo mensaje
           notifyAll();
         }

class consumidor extends Thread {
        productor elProductor ;
        String nombre;

        consumidor ( String pNombre, productor
        unProductor ) {
         elProductor = unProductor ;
         nombre = pNombre;
        }

        public void run () {
         try {
            while ( true ) {
                String cadMensaje= elProduc-
        tor.recibirMensaje();
                System.out.println ( nombre + "
        recibe " + cadMensaje );
                sleep( 2000 ); // Consumidor a
        dormir ...
            }
         }
         catch ( InterruptedException e ) { }

        }

        public static void main ( String args[] )
        {
         productor miProductor = new produc-
        tor();
         miProductor.start();
         new consumidor( "consumidor1",
        miProductor ).start();
         new consumidor( "consumidor2",
        miProductor ).start();
        }
}
```

La ejecución muestra la siguiente salida por pantalla tal y como la exponemos en el código que describimos a continuación:

```
consumidor1 recibe Sat Jun 27 19:56:40
      GMT+02:00 1998
consumidor2 recibe Sat Jun 27 19:56:41
      GMT+02:00 1998
consumidor1 recibe Sat Jun 27 19:56:42
      GMT+02:00 1998
consumidor2 recibe Sat Jun 27 19:56:43
      GMT+02:00 1998
```

Como se observa se alternan ordenadamente los dos consumidores como resultado de la llamada *sleep()* en los distintos métodos. Es un buen ejercicio comprobar qué sucede si se eliminan las llamadas a *sleep()*, ya que según la implementación Java que estemos utilizando observaremos diferentes resultados.

Además, se puede comprobar que no es necesaria modificación alguna sobre el método *crearMensaje()* del productor, es decir, la sentencia *notifyAll()* puede ser también *notify*() ya que si un consumidor está esperando a que aparezca un mensaje en la cola, no es posible que el productor este esperando simultáneamente porque la cola esté llena.

Como se ha detallado, el uso de *synchronized* y el par *notify/wait* permite a los *thread* un tratamiento correcto de los datos compartidos, es decir, una sincronización de acceso y tratamiento. Como hemos visto, *synchronized* resulta más conveniente para controlar el acceso simultaneo a un mismo método de un objeto. El par *notify/wait* permite a los *threads* una sincronización más definida, además de unos mecanismos de bloqueo basados en la espera y notificación.

UN THREAD ESPECIAL, EL THREAD "DAEMON"

Un *thread Daemon* (demonio) es un thread de baja prioridad que consta de un bucle infinito y se suele utilizar para prestar servicios cuando se necesite, por ejemplo, refresco de memoria, mostrar imágenes, etc. Java utiliza un *thread daemon*, el recolector de basura para realizar todas las gestiones sobre la creación y destrucción de los objetos en memoria.

Un *thread* demonio se declara de la siguiente manera:

```
Thread.setDaemon();
```

y se puede comprobar si un *thread* es tipo "demonio" mediante el siguiente método:

```
isDaemon();
```

La diferencia entre los *threads* normales y los demonios es que Java no espera la muerte de los demonios para detener su ejecución. Los *threads* normales impiden al núcleo de Java parar el programa principal hasta que estos no mueren bien por la llamada a *stop()* o por la terminación normal del *thread*. En el caso de los demonios, Java los elimina cuando comprueba que únicamente quedan *threads* de del tipo demonios. Su ventaja consiste en que no es necesario controlar su finalización, de tal forma que se evitan los clásicos procedimientos de pararCopiar(), detenerSpool(), etc.

Otra característica consiste en que cualquier thread que se crea desde otro que es *daemon* también es un *thread* demonio.

El siguiente ejemplo demuestra la creación de *threads* demonios:

```
import java.io.*;

class demonio extends Thread {
   private int MAXELEM = 10 ;
   private Thread [] t = new Thread [MAXE-
       LEM];
   public demonio () {
      setDaemon(true);
      start();
   }

   public void run() {
      for ( int i =0 ; i < MAXELEM ; i ++)
         t[i] = new demonioHijo(i);
      for ( int i =0 ; i < MAXELEM ; i ++) {
         System.out.println ( " t[ "+i+ "] is.Dae-
        mon() = " + t[i].isDaemon() );
      }
      while (true)  yield();
   }
}

class demonioHijo extends Thread {
   public demonioHijo ( int i ) {
      System.out.println ( " demonioHijo " + i +
         " iniciado ");
      start();
   }
   public void run() {
      while (true)  yield();

   }
}

public class misdemonios {
   public static void main ( String [] args) throws
       IOException
   {
      Thread d = new demonio();
      System.out.println ( "d.isDaemon() " +
       d.isDaemon()
         System.in.read();
);
}
```

CONCLUSIONES

Como se ha podido comprobar a lo largo de este artículo, el lenguaje Java nos permite unos sencillos y efectivos mecanismos de sincronización y comunicación entre *threads*, que van desde el uso de la palabra clave *synchronized* hasta el par *wait/notify*.

Gracias a estos métodos se puede controlar el desarrollo de aplicaciones que utilicen recursos compartidos entre múltiples *threads*. Además hemos aprendido a desarrollar los *threads* "demonios", tan comunes en lenguajes y sistemas operativos, con el fin de tener la posibilidad de ofrecer servicios a otros *threads*.

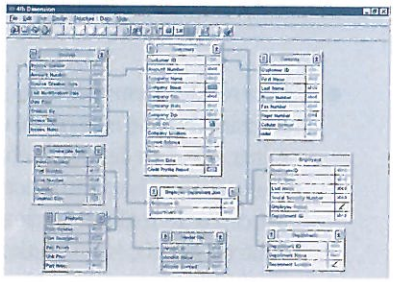
Si le gustaron otras bases de datos, 4th Dimension v6 *le sorprenderá*

4th Dimension es el entorno de bases de datos integrado más completo del mercado. Años de experiencia en el diseño de bases de datos han permitido satisfacer las siguientes necesidades:

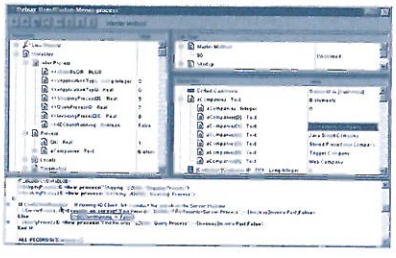
- Automatizar cualquier proceso.
- Diseñar modelos gráficos eficientes y orientados a objeto.
- Suministrar la más amplia gama de herramientas para facilitar el diseño y mantenimiento de bases de datos.
- Permitir una fácil reutilización de código.
- Permitir escalar desarrollos Cliente/Servidor, web y multiplataforma de forma transparente.

Estas son algunas de las **características** de 4th Dimension:

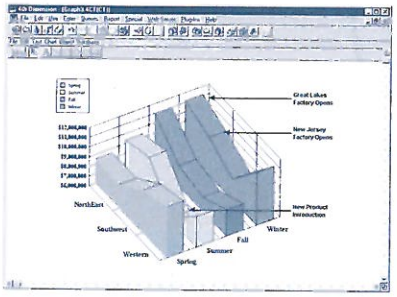
- **Motor de base de datos multiproceso, triggers, procedimientos almacenados**, presentación visual de tablas mediante diagrama de entidad /relación (ERD)



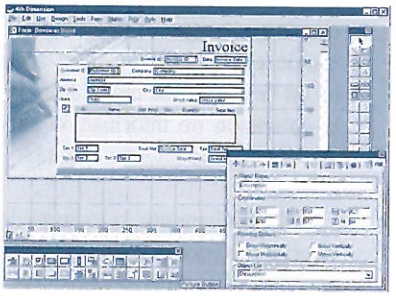
- **Debugger visual** multiproceso de última generación. **Intérprete** incluido para testeo y debugging.



- **Wizards avanzados** para diseño de formularios, queries, informes, importación, exportación, etiquetas, etc. Editor de gráficas programable 2D y 3D, soporta gráficas de barras, tartas, áreas, líneas, etc.

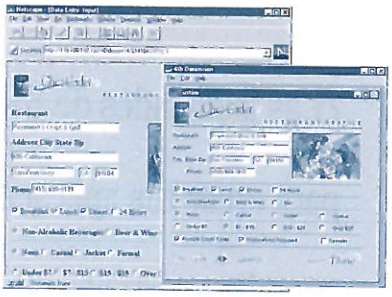


- **Editor de formularios basado en objetos.** Gestión de hojas de estilo. Redimensionamiento automático. **Docenas de objetos de interface** incluyendo check boxes, radio buttons, push buttons, áreas de scroll, picture button, botones invisibles, tab control, listas jerárquicas, termóme-tros, reglas, combo box, listas menú desplegables, menús pop-up jerárquicos, menús picture, radio pictures, listas desplegables, menús pop-up, subforms, group boxes, etc.

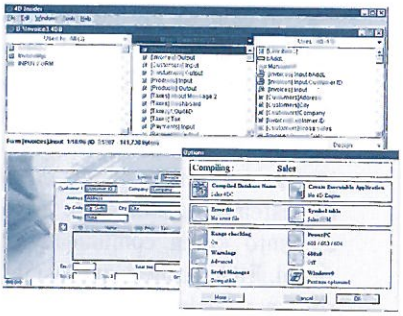


- Arquitectura de distribución de aplicaciones integrada y **soporte para proyectos con múltiples desarrolladores** con check-in/check-out automatizado.
- Arquitectura extensible mediante **plug-ins** y DLLs.

- **Un sólo código fuente** desplegable sin modificaciones de monousuario a 4D Server, la versión Cliente/Servidor de 4th Dimension.
- Soporte de **ODBC** (cliente y servidor) y conexiones nativas **SQL** a ORACLE, Sybase y MS SQL Server.
- **Servidor Web integrado** para acceso automático a formularios y gestión Cliente/Servidor mediante navegadores web.



- **Compilador a código máquina real** optimizado para diferentes procesadores. **Gestor gráfico de referencias cruzadas** y librerías de objetos.



VERSIÓN DEMO GRATUITA

4th Dimension es el núcleo de la gama de productos 4D. Visite http://www.ambitconsulting.com o llame al 93-209 41 11 para más información o para recibir una versión demo gratuita. Mencione este anuncio y reciba 4th Dimension v6 por el precio promocional de 34.900 Ptas*. (precio habitual: 79.000 Ptas.).

Balmes, 297, 4º, 2ªb. 08006 Barcelona
t. | (34) 93 209 41 11 f. | (34) 93 240 18 88
e-mail: ambit@ambitconsulting.com
http://www.ambitconsulting.com

AMBIT Consulting

(* Actualización competitiva. Oferta válida hasta el final de existencias)

Más allá de SVGALIB: VGAGL (III)

Jorge F. Delgado Mendoza (ernar@convex.es)

¿Las facilidades de SVGALIB no son suficientes?. ¿Necesitas capacidades avanzadas para crear tus aplicaciones?. VGAGL es la respuesta: BitBlt Transfer, Page Flipping, pantallas virtuales, Fast Clipping, son sólo algunas de las capacidades que esta librería añade a las primitivas de SVGALIB, completando así su funcionalidad.

A lo largo de los dos artículos previos dedicados a la librería gráfica de Linux llamada SVGALIB hemos ido aprendiendo un gran número de aspectos relativos a la programación gráfica, dentro de uno de los sistemas operativos de mayor crecimiento en la comunidad internacional. En la primera entrega de la serie, un artículo completamente teórico, describimos el procedimiento de instalación de la librería. Poco a poco fuimos dando todos los pasos necesarios para poder utilizar el software en nuestra máquina Linux.

También se comentó previamente de una forma breve la arquitectura de SVGALIB, de tal manera que el lector se fuera familiarizando con sus procedimientos y particularidades. De esta manera conseguíamos acostumbrar al lector a las ideas que más tarde se expondrían en los diferentes ejemplos.

Para acabar el artículo, describimos un conjunto muy básico de las funciones que proporciona la librería, centrando el aprendizaje en la activación de un determinado modo de vídeo y en la obtención de información sobre las capacidades de nuestra tarjeta.

Paralelamente, a lo largo de todo el primer artículo fuimos introduciendo conceptos relacionados con la arquitectura del subsistema de vídeo del PC. Este conocimiento resulta imprescindible a la hora de entender el segundo de los artículos que componen la serie, donde se describieron dos conceptos fundamentales: la arquitectura de la memoria de vídeo y la diferencia entre color directo e indexado. A continuación se describieron, en profundidad y con varios ejemplos, las primitivas de SVGALIB. Para una mejor comprensión, se agruparon en cuatro grupos:

- Control de Memoria: este grupo de primitivas se encarga de determinar qué tipo de arquitectura de memoria se encuentra activado, así como de realizar funciones de control de acceso - rutinas de cambio de banco, etc.
- Manejo de paleta: estas funciones permiten controlar en todo momento la asignación de colores de nuestros dibujos.
- Dibujo: diseñadas para facilitar la tarea del programador, evitando que tenga que realizar formas simples punto a punto.
- Acceso Directo a Memoria: permite obtener la mayor velocidad posible de la tarjeta de vídeo.

En este artículo completaremos el estudio de las primitivas gráficas,

dejando el manejo de otros periféricos para el último número de la serie. Para ello vamos a comenzar con una breve reseña de las posibilidades que ofrece VGAGL a la hora de crear aplicaciones gráficas. Una vez acabada, entraremos de lleno en una descripción detallada de las diferentes rutinas disponibles.

VGAGL introduce dos conceptos radicalmente nuevos: ventanas virtuales y contextos de trabajo

Para tener una idea más clara acerca de los componentes de la librería, dividiremos nuestro estudio en cuatro partes, que incluyen aquellas funciones de características similares.

Los diferentes grupos de funciones son los siguientes:

- Manejo de Contextos: aquí podremos encontrar todo tipo de funciones asociadas al manejo de múltiples pantallas gráficas.
- Dibujo: dentro de esta sección agruparemos todas aquellas rutinas que permiten crear formas simples: círculos, líneas, etc.
- Recortado: son todas aquellas funciones especiales dedicadas a crear zonas de exclusión en la pantalla.
- Otras funciones: en este grupo deberemos incluir todas aquellas primitivas que, sin pertenecer a un grupo definido, son lo suficientemente útiles como para que merezca la pena indicarlas.

A lo largo de todo el artículo, iremos ofreciendo ejemplos que el lector podrá encontrar, como viene siendo habitual, en el CD-ROM que acompaña a la revista.

Comenzamos, pues, la tercera entrega de la serie enumerando los requisitos necesarios para que el lector pueda aprovechar toda la información que se expone a lo largo de las próximas páginas.

REQUISITOS

CONOCIMIENTOS

Como en todos los artículos anteriores, es necesario un conocimiento relativamente avanzado de lenguaje de programación C para poder sacar el máximo partido del artículo. Sin embargo, un conocimiento general de programación imperativa será suficiente para que el lector sea capaz de asimilar los conceptos que se exponen es esta entrega, ya que se ha intentado no hacer uso de ninguna característica exclusiva del lenguaje.

Incorpora funcionalidades que hacen mucho más fácil el desarrollo de una aplicación

A la hora de desarrollar aplicaciones la situación es algo diferente. Ya que la mayoría del código disponible para Linux está desarrollado en C, la mejor manera de construir programas útiles accediendo a las librerías del sistema es empleando ese mismo lenguaje.

Para mantener el control de dependencias entre los diferentes ficheros, así como para generar ejecutables utilizaremos la herramienta GNU Make. Un conocimiento de la sintaxis asociada a sus ficheros de configuración, también llamados makefiles, nos va a permitir realizar todas las tareas asociadas a la creación de ejecutables de una manera totalmente automática.

El compilador que se va a utilizar para todos los ejemplos es el GNU C Compiler comúnmente llamado gcc. Un conocimiento, aunque sólo sea superficial, de las diferentes opciones de compilación puede ayudar mucho en el seguimiento y depuración de errores en nuestro código.

Finalmente, si el lector está familiarizado con el sistema gráfico del PC, encontrará el artículo mucho más fácil de seguir. En las dos entregas anteriores hemos aprendido algunos de los conceptos más importantes relativos a este tema. De esta forma siglas como bpp o RGB no deberían asustar al lector que haya seguido la serie completa de artículos. Del mismo modo, también estará al tanto de los modelos de direccionamiento de la memoria, de la diferencia entre color directo y color indexado, etc.

SOFTWARE

Necesitaremos tener instaladas las siguientes herramientas:

- gcc 2.7.0 o superior.
- make.
- SVGALIB 1.2.13.

Todos los ejemplos que se presentan en este artículo han sido realizados utilizando las siguientes versiones de las anteriores herramientas:

- gcc versión 2.7.2.3
- GNU Make versión 3.76.1
- SVGALIB versión 1.2.13

DOCUMENTACIÓN

Aunque no es necesario utilizar ningún tipo de información adicional a los artículos, cualquier lector que desee aumentar sus conocimientos en el campo de la programación gráfica puede acudir a las siguientes fuentes:

- Para detalles de la programación de tarjetas de vídeo, así como de los aceleradores gráficos, puede acudir al excelente libro que sobre el tema ha publicado Richard F. Ferraro, llamado Programmer's guide to SVGA programming. Aunque el libro está escrito en inglés, sin ninguna duda es la biblia en lo que a tarjetas de vídeo se refiere. Este texto está editado por Addison-Wesley.

- Para que el lector interesado pueda conseguir información relativa a la programación gráfica desde un punto de vista orientado a la creación de aplicaciones, os remitimos a la siguiente URL :

http://www.cs.umu.se/~christer/GR/development.html

A lo largo del artículo se van a describir numerosas funciones de la librería. En muy pocos casos se va a dar una sintaxis exacta de la función, ya que es preferible dar al lector una idea clara de lo que va a obtener con cada una de ellas. Si en algún caso fuera necesario obtener esta información, bastará con utilizar las páginas de manual correspondientes a la rutina que se desee utilizar. El acceso a las páginas de manual, directo una vez hayamos instalado la librería, se realiza mediante el comando siguiente:

man <nombre de la función>

Los procedimientos de inicialización de SVGALIB siguen siendo necesarios, ya que VGAGL no se encarga de estas tareas

HARDWARE

Aunque no se ha comentado en artículos anteriores, tener una tarjeta de vídeo soportada por SVGALIB es muy interesante, ya que nos va a dar la posibilidad de utilizar un mayor número de modos y resoluciones. En el caso de no tener una tarjeta soportada por SVGALIB, tendremos que conformarnos con los modos de vídeo VGA estándar. Estos modos tienen su tope en 640x480 para 16 colores y 320x200 en 256 tonos. Es imposible utilizar 15/16 bpp ó 24/32 bpp con una tarjeta que no esté soportada por la librería SVGALIB.

¿PARA QUÉ SIRVE VGAGL?

Se trata de una buena pregunta, puesto que a primera vista podríamos pensar que con todo lo que hemos aprendido en los dos artículos anteriores es posible crear cualquier tipo de operación gráfica. Sin embargo, VGAGL incorpora una serie de funcionalidades que van a hacer mucho más fácil el desarrollo de una aplicación. Además incorporara un elevado número de primitivas nuevas, donde la mayoría se encuentran destinadas a la presentación de imágenes en pantalla o a realizar dibujos simples. En este sentido la librería VGAGL introduce dos conceptos radicalmente nuevos que son los siguientes:

- Pantalla Virtual: la librería nos va a permitir reservar una zona en la memoria principal de nuestro ordenador, organizada de tal manera que vamos a poder utilizarla como si tratase de la memoria de la tarjeta de vídeo.

- Contexto Gráfico: se trata de unas estructuras especiales que permiten definir una serie de parámetros, aplicables únicamente cuando un determinado contexto se encuentre activo.

Gracias a la combinación de estas dos ideas, se podrán crear efectos que, de otra manera, hubieran requerido una gran cantidad de esfuerzo por parte de programador.

MANEJO DE CONTEXTOS

Antes de realizar ningún tipo de operación con la librería VGAGL, es necesario definir, al menos, un contexto de operación. Este procedimiento de inicialización se debe realizar a continuación de las rutinas de arranque de SVGALIB, ya que VGAGL no se encarga de estas tareas. A la hora de definir contextos, tenemos dos posibilidades:

- Podemos crear contextos sobre la memoria de vídeo de la tarjeta de vídeo.

- Podemos utilizar la memoria principal del PC para realizar la manipulación gráfica, definiendo pantallas virtuales de vídeo.

Una tarjeta de vídeo soportada por SVGALIB posibilita la utilización de un mayor número de modos y resoluciones

En toda aplicación que utilice VGAGL tendremos que definir al menos un contexto sobre la memoria gráfica de la tarjeta. Es posible definir más de uno, si queremos utilizar modos de diferentes características a lo largo de nuestra aplicación.

A primera vista, podríamos cuestionarnos la utilidad de las pantallas virtuales en la memoria del ordenador. Sin

Listado 1. Ejemplo 1.C.

```
#include <stdio.h>
#include <vga.h>
#include <vgagl.h>


void
main()
{
   char c;
   vga_modeinfo * info;
   int x;
   int y;
   GraphicsContext physicalscreen,
memoryscreen;

   vga_disabledriverreport();
   vga_init();

   vga_setmode(G320x200x256);
   /* Creamos un contexto en la pan-
talla de la memoria de vídeo */

   gl_setcontextvga
(G320x200x256);

   gl_getcontext(&physicalscreen);


   gl_setcontextvgavirtual
(G320x200x256);
   gl_getcontext(&memoryscreen);

      /* Colocamos un color azul en
el índice 10 */

   gl_setpalettecolor ( 10 , 0, 0, 20 );

   /* Rellenamos toda la pantalla con
ese color */

   gl_clearscreen( 10 );

   /* la copiamos a la pantalla princi-
pal */

   gl_copyscreen ( &physicalscreen);


   while ( 1 )
      {
            c = getchar();
            if ( c == 'q' )
                  break;
      }
   vga_setmode(TEXT);

}
```

embargo, si nos paramos a pensar en sus potenciales utilizaciones, podemos encontrar rápidamente algunos usos:

- En aplicaciones que creen una imagen lo suficientemente compleja como para que se pueda ver cómo se dibuja en la pantalla, podemos realizar todas las operaciones en memoria principal y luego trasladarla instantáneamente a la memoria de vídeo.
- Podemos utilizar estas pantallas virtuales a modo de almacén para datos ya dibujados, ahorrándonos la necesidad de volverlos a crear.
- En caso de que utlicemos la extensión de X-Windows conocida como MITSHM (Memoria Compartida), podemos portar una aplicación SVGALIB a X-Windows de una manera totalmente directa.

Gracias a VGAGL, nuestras aplicaciones pueden realizar animaciones suaves y precisas, utilizando la técnica conocida como doble o triple buffering

Estas son sólo algunas de las posibilidades que nos brinda el empleo de ventanas virtuales, sin duda un programador experimentado podrá aplicar esta herramienta para sus propias aplicaciones. Las funciones que permiten manipular los diferentes contextos son las expuestas a conntinuación:

- gl_getcontext()
- gl_setcontext()
- gl_setcontextvga()
- gl_setcontextvgavirtual()
- gl_allocatecontext()
- gl_freecontext()
- gl_setcontextwidth()
- gl_setcontextheight()

A continuación describimos en detalle cada una de las rutinas mencionadas. En el fichero ejemplo1.c (incluido en el CD-ROM) se pueden encontrar detalles relativos al manejo de estos diferentes contextos.

FUNCIÓN GL_SETCONTEXTVGA()

Esta función es la encargada de colocar el contexto actual de acuerdo a los parámetros del modo de vídeo VGA que se le pasa como argumento. El contexto utilizará directamente la memoria de vídeo para realizar las diferentes operaciones. Antes de poder utilizar las primitivas de dibujo, es necesario activar el modo mediante una llamada a vga_setmode().

FUNCIÓN GL_SETCONTEXTVGAVIRTUAL()

De un modo similar a la función anterior, gl_setcontextvgavirtual lo que crea es un contexto cuyas características se corresponden con las del modo de vídeo que se le pasa como argumento. Sin embargo, la memoria que se va a utilizar para guardar la información de este contexto se encuentra en la RAM principal del PC y no en la memoria de vídeo. Para que podamos visualizar los datos contenidos en la pantalla virtual, tendremos que copiar su contenido a la memoria de la tarjeta de vídeo del PC, mediante la función gl_copyscreen().

FUNCIÓN GL_SETCONTEXT()

Esta función sirve para activar un contexto determinado. A partir de ese momento, todas las rutinas de dibujo pasan a ejecutarse sobre la memoria asociada al mismo. No copia los contenidos de un contexto a otro, para realizar este tipo de tareas existen otras funciones de la librería que se ocupan de ello, entre las que se incluyen: gl_copyscreen(), gl_copyboxtocontext(), gl_copyboxfromcontext().

Listado 2. Ejemplo 2.C.

```
#include <stdio.h>
#include <vga.h>
#include <vgagl.h>


void
main()
{
   char c;
   vga_modeinfo * info;
   int x;
   int y;
   GraphicsContext physicalscreen;


   vga_disabledriverreport();
   vga_init();

   vga_setmode(G320x200x256);

   /* Creamos un contexto en la pantalla
de la memoria de vídeo */

   gl_setcontextvga (G320x200x256);

   gl_getcontext(&physicalscreen);

   /* Colocamos un color azul en el
índice 10 */

   gl_setpalettecolor ( 10 , 0, 0, 20 );

   /* Rellenamos toda la pantalla con
ese color */

   gl_clearscreen( 10 );

   /* Colocamos un rojo en la posición
11 */

   gl_setpalettecolor ( 11 , 63, 0, 0 );

   /* Dibujamos un punto de ese color
en las coordenadas (100,100) */

   gl_setpixel ( 100 , 100 , 11 );

   /* ahora dibujamos una línea horizon-
tal */

   gl_hline ( 50 , 50 , 150 , 11 );

   /* y ahora completamos hasta conse-
guir un cuadrado */

   gl_hline ( 50 , 150 , 150 , 11 );

   gl_line  ( 50 , 50 , 50 , 150 , 11 );
   gl_line  ( 150 , 50 , 150 , 150 , 11 );


   /* Colocamos un gris en la posición
12 */

   gl_setpalettecolor ( 12 , 40, 40, 40 );

   gl_circle ( 100, 100, 25, 12);

   /* ahora creamos un rectángulo con el
mismo color */

   gl_fillbox ( 200, 150 , 50 , 35 , 12 );


   while ( 1 )
      {
         c = getchar();
         if ( c == 'q' )
            break;
      }
   vga_setmode(TEXT);

}
```

FUNCIÓN GL_GETCONTEXT()

Imagen especular de la anterior, esta función es la que se encarga de guardar toda la información asociada al contexto actual en la estructura que el usuario elija.

Con estas cuatro funciones descritas en estas líneas ya podemos realizar cambios de contexto de uno a otro lado. El siguiente fragmento de código que detallamos a continuación se puede utilizar para crear dos contextos, uno virtual y otro sobre la memoria de vídeo, y activar uno u otro:

```
GraphicsContext pantalla, memoria;
gl_setcontextvga(G320x200x256);
gl_getcontext (&pantalla);
gl_setcontextvgavirtual(G320x200x256);
gl_getcontext(&memoria);
```

Para conmutar de uno a otro, bastará simplemente con que utilicemos las siguientes llamadas:

```
gl_setcontext(&pantalla);
gl_setcontext(&memoria);
```

FUNCIÓN GL_SETCONTEXTVIRTUAL()

Si queremos modificar o actualizar los parámetros más importantes de un modo, podemos usar esta rutina, mediante la que podremos alterar los siguientes valores:

> Gl setcontex() se utiliza para activar un contexto determinado que a partir de entonces ejecuta las rutinas del dibujo sobre la memoria asociada del mismo

- Anchura en puntos de la pantalla.
- Altura en puntos de la pantalla.
- Número de bytes por punto.
- Número de bytes, con información de color, por punto.
- Dirección de la memoria de vídeo del contexto.

Las demás funciones asociadas al manejo de contextos sirven para distintos aspectos como liberar estructuras en el caso de - gl_freecontext(), - reservar espacio para las mismas como, - gl_allocatecontext(), - o determinar propiedades del modo, - gl_setcontextwidth() y gl_setcontextheight().

DIBUJANDO

Las nuevas primitivas de dibujo se solapan con algunas de las que ya hemos visto. A diferencia de las que ya conocíamos, cuyas operaciones se realizaban directamente en memoria de vídeo, todas las rutinas de VGAGL operan sobre el contexto que se encuentre activo.

En todos los casos VGAGL se encarga de realizar las operaciones de cambio de banco que sean necesarias sdfsdfsdf

El segundo de los ejemplos (ejemplo2.c) ilustra la utilización de varias de las funciones que se describen a continuación. En todos los casos, VGAGL se encarga de realizar las operaciones de cambio de banco necesarias. Esta tarea provoca una sobrecarga apreciable, por lo que se recomienda utilizar un mapa lineal de memoria si es posible.

FUNCIÓN GL_CLEARSCREEN()

Esta función es la encargada de borrar la pantalla asociada al contexto seleccionado. Los datos asociados a esta pantalla virtual pueden encontrarse tanto en memoria del sistema como directamente sobre la tarjeta de vídeo. Toma como único argumento el color con el que vamos a borrar la pantalla virtual.

FUNCIÓN GL_RGBCOLOR()

Se utiliza para obtener el valor asociado a un color determinado en el contexto activo. Los datos RGB del tono seleccionado se ofrecen como argumento a la función. En el siguiente ejemplo utilizamos gl_rgbcolor y gl_clearscreen para rellenar la pantalla de color blanco:

```
gl_clearscreen ( gl_rgbcolor ( 255, 255, 255 ) );
```

Esta función sólo tiene sentido en los modos de color directo, puesto que en el caso de color indexado (256 colores), podemos llegar a crear un sistema de color pseudo-directo con la función llamada: gl_setrgbpalette().

FUNCIÓN GL_SETPALETTECOLOR(),

Una llamada a esta función permite asociar un color determinado a un índice de la tabla de colores. Como todas las demás funciones que están relacionadas con el manejo de paletas sólo tiene sentido en modos de color indexado (256 colores).

En el caso de poder hacer algo de dos maneras diferentes, siempre podremos conseguir mayor velocidad con la más restrictiva de las dos. gl_hline() y gl_line() son un ejemplo claro

En caso de que queramos colocar el color blanco en el índice número 10 de la tabla de colores, utilizamos la siguiente llamada:

```
gl_setpalettecolor( 10 , 63, 63, 63 );
```

Utilizando la tabla de colores, podríamos escribir el ejemplo de la función anterior de la siguiente manera que exponemos:

Tabla 1. Componentes RGB, en función del índice seleccionado, para el modo pseudo-directo de 8 bits creado por la función gl_setrgbpalette().

Color base	Bits del índice
Rojo (R)	6-7
Verde (G)	3-5
Azul (B)	0-2

```
gl_setpalettecolor( 10 , 63, 63, 63 );
gl_clearscreen ( 10 );
```

FUNCIÓN GL_GETPALETTECOLOR()

Se trata de una función similar a la anterior con la diferencia de que en vez de colocar un color en un índice determinado, devuelve el valor RGB asociado al índice que le pasamos como parámetro.

Además de estas dos funciones, existen otras cuatro muy similares en funcionamiento a sus equivalentes en SVGALIB. Estas funciones son: gl_setpalettecolors(), gl_setpalette(), gl_getpalettecolors() y gl_getpalette(). Permiten manipular varios índices de la tabla de colores a la vez de una manera muy similar a las dos funciones expuestas.

Para una descripción en detalle de su funcionamiento, se remite al lector a las páginas de manual de cada una de ellas.

FUNCIÓN GL_SETRGBPALETTE()

Esta función nos permite utilizar un modo de color indexado como si fuera de color directo. Para ello es capaz de crear una paleta especial en la que cada índice tiene sus propios componentes RGB.

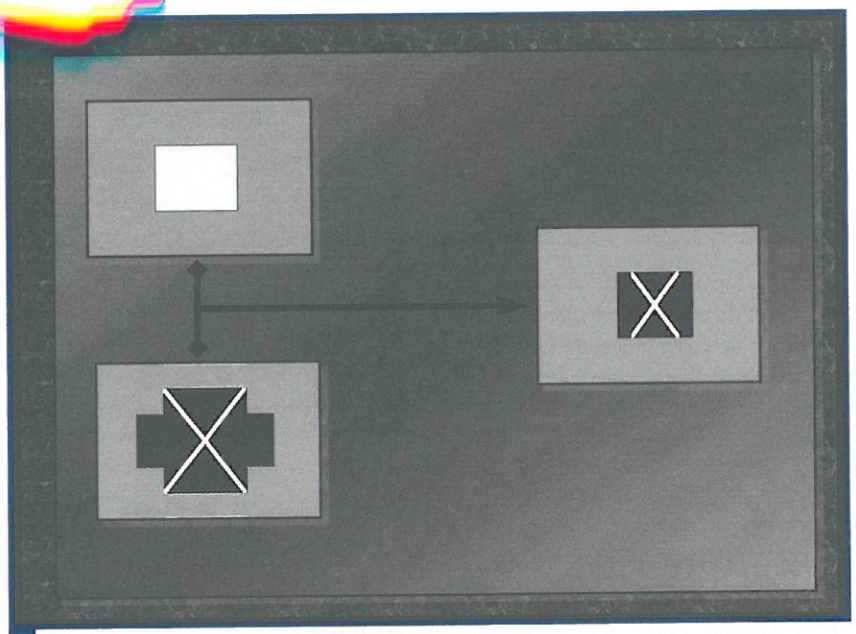


Figura 1. La técnica conocida como Clipping permite definir áreas, más allá de las cuales no se pueden modificar puntos. Al definir un rectángulo de acotación, lo que en realidad estamos haciendo es crear una máscara AND en la cual sólo están activados los puntos interiores al rectángulo.

FUNCIÓN GL_SETPIXEL()

Esta función permite dibujar un único punto en el contexto activo. Los dos primeros argumentos son las coordenadas x e y del punto, mientras que el tercero es el valor del color. Este valor tiene distintos significados en función de los colores del contexto activo:

- Color Indexado (256): sólo se toman los 8 bits menos significativos, utilizándose como índice para la paleta de colores activa.
- Color Directo (16 bit): los 16 bits menos significativos se utilizan para determinar las componentes RGB del color seleccionado.
- Color Directo (24/32 bit): los 24 bits menos significativos se toman como valores RGB del color a utilizar.

Existe una función similar llamada gl_setpixelrgb() a la cual se le pasan directamente las componentes RGB del color que se va a utilizar. En los modos de 256 colores esta primitiva sólo tiene sentido si se ha llamado previamente a gl_setrgbpalette().

FUNCIÓN GL_HLINE

Con gl_hline() podemos dibujar una línea horizontal en el contexto activo de la manera más rápida posible. Debido a que realiza determinadas suposiciones sobre la arquitectura del PC, - y muchas veces se apoya en el acelerador de vídeo, - utilizar esta función supone un gran ahorro de tiempo sobre la rutina genérica de líneas. En ningún caso deberemos dibujar líneas a base de puntos, ya que perderemos varios órdenes de magnitud en la velocidad de ejecución.

Con gl_hline() podemos dibujar una línea horizontal en el contexto activo de la manera más rápida posible

FUNCIÓN GL_LINE()

Consiste en la función genérica de creación de líneas. A pesar de no ser tan rápida como la función gl_hline() permite crear vectores en cualquier dirección. Si se pretende dibujar un cuadrado de cincuenta puntos de lado y cuya esquina superior izquierda esté situada en el origen de coordenadas, podríamos utilizar la siguiente secuencia de instrucciones:

```
gl_hline ( 0 , 0 , 50 , color );
gl_hline (0 , 50 , 50, color );
gl_line ( 0 , 0 , 0 , 50 , color );
gl_line (50, 0 , 50 , 50, color);
```

Por supuesto, esta secuencia de instrucciones supone que hemos definido un color de primer plano y lo hemos introducido en la variable color. Es necesario hacer notar que, debido a que el algoritmo de creación de líneas está basado en el modelo de Bressenham, es probable que si intercambiamos los puntos origen y destino de una línea oblicua el resultado final sea distinto.

La mayoría de las primitivas de VGAGL sólo tienen sentido dentro del contexto activo

FUNCIÓN GL_CIRCLE()

¿Estamos ya cansados de líneas y puntos?, está bien, gl_circle() permite dibujar un círculo centrado en un par de coordenadas relativas al origen del contexto actual, con un radio y un color determinados, esta función es mucho más lenta que la generación de líneas si comparamos el número de puntos que se pueden dibujar por unidad de tiempo. Si nuestra tarjeta tiene un acelerador soportado por SVGALIB, el tiempo necesario para crear los círculos se reducirá de forma notable. La siguiente línea crea un círculo de vein-

ticinco puntos de radio, centrado en las coordenadas (100,100):

```
gl_circle ( 100, 100, 25, color );
```

FUNCIÓN GL_FILLBOX()

La última de las rutinas básicas que ofrece VGAGL para la creación de dibujos es gl_fillbox(). Esta primitiva acepta cinco argumentos los cuales indican respectivamente la coordenada superior izquierda, la anchura, la altura y el color del rectángulo que se quiere dibujar. En el ejemplo número 2 del CD se pueden ver algunos usos, comentados, de todas estas funciones.

RECORTANDO

Hay veces que dibujar sobre una pantalla no es suficiente, sino que en ocasiones nos hemos encontrado con la necesidad de mostrar sólo parte de nuestra imagen, limitándola a un área determinada.

> Gl_copyscreen() tiene la misión de copiar el contenido de la memoria del contexto actual en otro diferente

VGAGL logra definir un rectángulo, mas allá de los bordes del cual no se va a modificar ningún punto. Por lo tanto, nuestro dibujo quedará truncado en aquellas zonas en las que sobresalga del paralelogramo definido.

En la jerga gráfica, esta operación se denomina clipping. La figura número 1, ilustra el método de funcionamiento de esta técnica a la que nos referimos. Las funciones que ofrece VGAGL para realizar acotado de bordes se describen a continuación.

FUNCIÓN GL_DISABLECLIPPING()

Una llamada a esta función desactiva la ventana de acotación. Este es el comportamiento por defecto de la librería. Si deseamos realizar dibujos acotados, deberemos activar la ventana mediante una llamada a gl_enableclipping().

FUNCIÓN GL_ENABLECLIPPING()

Para poder utilizar la ventana de acotación deberemos activar el modo de clipping, al tiempo que definimos la ventana que vamos a utilizar. Una vez invocada esta primitiva, el sistema acota automáticamente todas las operaciones de dibujo que encuentre. Para desactivar esta función, debemos llamar a gl_disableclipping().

FUNCIÓN GL_SETCLIPPINGWINDOW()

Con una llamada a esta rutina, definimos las esquinas superior izquierda e inferior derecha del rectángulo que va a servir de máscara para nuestro dibujo. Esta función siempre debe ser llamada después de invocar gl_enableclipping(). Si quisiéramos dibujar la parte de circunferencia inscrita en un rectángulo cuyo lado es igual al 75% del diámetro de la misma, utilizaríamos la siguiente secuencia de órdenes:

```
gl_enableclipping();
gl_setclippingwindow( x, y ,
x+(0.75*diametro) , y+(0.75*diametro));
gl_circle ( x + (0.375*diametro) , y +
(0.375*diametro) , diametro, color);
gl_disableclipping();
```

Gracias a estas tres primitivas con las que trabajaremos, tenemos la posibilidad de realizar complejas combinaciones de ventanas e imágenes, sin que unas lleguen a solaparse con otras a la hora de dibujarlas.

OTRAS FUNCIONES

Además de las primitivas que hemos descrito a lo largo de las tres secciones anteriores, existen algunas funciones cuya importancia nos obliga a incluirlas en el artículo. Aunque su propósito es variado, todas ellas están relacionadas con el manejo de imágenes en la pantalla.

> VGAGL nos ofrece la posibilidad de generar máscaras rectangulares para acotar los datos que enviamos a la pantalla

FUNCIÓN GL_COPYSCREEN()

Aunque ya hemos visto como se utiliza esta función, hasta ahora no la habíamos estudiado en toda su profundidad. La misión de la que se encarga es la de copiar el contenido de la memoria del contexto actual en otro contexto, - por ejemplo la pantalla física, - que se especifica en la llamada.

Una muestra muy ilustrativa de la utilización y propiedades de esta función se puede encontrar dentro del CD de la revista, incluido en el fichero ejemplo1.c.

FUNCIÓN GL_COPYBOX()

Con esta función podemos copiar un área rectangular de la pantalla a

otra posición dentro de la misma. no borra la zona de origen, sino que se limita a obtener los datos que allí residen y colocar una imagen idéntica en otra posición.

Esta función es lo que en la jerga gráfica se denomina ScrtoScr copy o BitBlt transfer. Por supuesto, la primitiva gl_copybox(), solo tiene sentido dentro del contexto que esté activo en ese momento. En el fichero ejemplo3.c podemos encontrar muestras de su utilización.

FUNCIÓN GL_COPYBOXFROMCONTEXT()

De una manera similar a la función anterior, esta primitiva permite copiar en el contexto activo un rectángulo de puntos. A diferencia de gl_copybox(), esta función va a buscar los datos originales al contexto cuyo identificador se le pasa como primer parámetro. En todos los demás aspectos, la función es idéntica a gl_copybox().

La función gl_putbox() se utiliza para colocar una imagen almacenada en la memoria del ordenador en el contexto actual

Existe una rutina gemela de gl_copyboxfromcontext(), llamada gl_copyboxtocontext(). Esta primitiva lo que nos permite enviar un rectángulo del contexto actual al que se le indica como destino. Sin embargo, la función gl_copyboxtocontext() es mucho más lenta en su ejecución que su hermana gemela, por lo que es bastante más aconsejable la utilización de esta última.

FUNCIÓN GL_PUTBOX()

Esta llamada se utiliza para colocar una imagen, almacenada en la memoria del ordenador, en el contexto actual. Esta imagen deberá estar ordenado de izquierda a derecha y de arriba a abajo en la estructura de datos correspondiente. Su tamaño deberá ser igual al ancho de la imagen por su altura, multiplicado por el número de bytes por punto del modo activado. Para obtener el dato asociado a los bpp de la imagen, podemos utilizar la macro global BYTESPERPIXEL. Su contenido es siempre el número de bytes que se necesitan para completar un punto del modo cuyo contexto está activo. Esta función realiza lo que en el mundo de los gráficos se denomina: MemtoScr copy.

Gl_copyboxfromcontext() permite copiar en el contexto activo un rectángulo de puntos

FUNCIÓN GL_SCALEBOX()

Con esta función vamos a completar nuestro recorrido por las primitivas que nos ofrece VGAGL. De manera similar a las demás funciones que operan con imágenes, esta función se encarga de manipular un rectángulo de datos. Sin embargo, a diferencia de las demás funciones que hemos estudiado, los polígonos de origen y destino de los datos son de distinto tamaño. Esta primitiva acepta como argumento un rectángulo de tamaño w por h almacenado en una posición de memoria determinada. Este argumento se escala apropiadamente hasta convertirlo en otro rectángulo de anchura y altura arbitrarias w1 y h1. Como en todas las funciones de VGAGL, se asume que el tamaño del punto es el asociado al contexto que esté activo en ese momento.

CONCLUSIONES

En estos tres artículos que hemos visto a lo largo de esta serie, dedicados al estudio de la librería gráfica de Linux llamada SVGALIB, hemos aprendido diferentes maneras de realizar nuestras propias aplicaciones gráficas. En el primer y segundo artículo, realizábamos manipulaciones directas en la memoria de vídeo gracias a las primitivas que nos proporcionaba SVGALIB. En el tercer artículo hemos ido más allá en nuestro estudio, incorporando a nuestros conocimientos casi todo el contenido de la librería VGAGL. Gracias a ello, hemos aprendido a realizar efectos especiales como cambios de página, cambios de contexto, generación de imágenes y un largo etcétera. En la próxima, - y por fin última, - entrega de esta serie vamos a aprender a manejar ciertas capacidades de la librería que no se utilizan directamente para la generación de imágenes.

En cualquier aplicación moderna, es imprescindible el uso de dos periféricos muy comunes: el teclado y el ratón.

Con VGAGL hemos logrado crear efectos especiale

En el número que cierra este ciclo de artículos estudiaremos el manejo de éstos desde dentro de la librería, así como la utilización de alguna primitiva avanzada de dibujo.

(1)RGB: acrónimo de las palabras inglesas Red, Green y Blue. Teóricamente, cualquier tono se puede representar como una combinación de estos tres colores. Describir un tono determinado en notación RGB consiste en cuantificar los pesos que corresponden a cada uno de los componentes.

Programación básica en Windows (III)

Jorge Figueroa (erde@arrakis.es)

A lo largo del artículo se explicará detenidamente la forma de programar uno de los recursos más usados en Windows: las barras de desplazamiento, las cuales, además de fáciles de crear, son un elemento muy importante a la hora de realizar aplicaciones capaces de manejar gran cantidad de información.

En el anterior capítulo de este pequeño curso de C quedó explicado a fondo el código fuente de un sencillo ejemplo de programa en C para Windows que poseía todos los elementos básicos. Ahora que se supone el lector ha quedado ya familiarizado con la forma en que se programa bajo este sistema operativo, se va a pasar a detallar cómo funcionan y cómo se programan las llamadas barras de desplazamiento, las cuales, son un recurso muy usado en este sistema operativo y que como se verá, son relativamente fáciles de entender y de aplicar. De esta forma se intentarán asentar los conocimientos adquiridos por el lector en los artículos anteriores.

Las barras de desplazamiento son uno de los recursos más usados de Windows

BARRAS DE DESPLAZAMIENTO

Como casi todos lo usuarios de aplicaciones saben por experiencia propia, son uno de los recursos más utilizados en Windows. Su utilidad reside en que ofrecen al usuario la posibilidad de acceder a gran cantidad de más información de la que cabe en la ventana de la aplicación. Es decir, cuando los datos sobrepasan el límite físico de las dimensiones de la ventana, las barras de desplazamiento permiten el desplazamiento por toda la información almacenada.

Como apreciaremos más adelante, las aplicaciones se pueden programar para que usen de forma individual las barras verticales, las horizontales o hacer que estén presentes ambas simultáneamente.

Como concepto relacionado, decir también que el cuadro de desplazamiento de una barra es el rectángulo que aparece dentro de la propia barra de desplazamiento, y se usa para que el usuario conozca la posición exacta de la ventana virtual que se está viendo respecto a la ventana real imaginaria que contiene toda la información realmente disponible.

CONSTANTE Y COMANDOS

Los comandos y constantes que se utilizan para manejar las barras se

Tabla 1. Constantes y comandos asociados a las barras de desplazamiento.

```
// Constantes:
#define SB_HORZ                 0
#define SB_VERT                 1
#define SB_TCL                  2
#define SB_BOTH                 3

// Comandos:
#define SB_LINEUP               0
#define SB_LINELEFT             0
#define SB_LINEDOWN             1
#define SB_LINERIGHT            1
#define SB_PAGEUP               2
#define SB_PAGELEFT             2
#define SB_PAGEDOWN             3
#define SB_PAGERIGHT            3
#define SB_THUMBPOSITION        4
#define SB_THUMBTRACK           5
#define SB_TOP                  6
#define SB_LEFT                 6
#define SB_BOTTOM               7
#define SB_RIGHT                7
#define SB_ENDSCROLL            8
```

encuentran definidos en el fichero *winuser.h*. Mediante ellos resulta más intuitivo y fácil el desarrollo de aplicaciones, evitando el manejo de valores sin relación alguna con lo que representan. En la tabla 1 se muestra una lista de dichos valores.

RANGO DE DESPLAZAMIENTO

Con este término nos estamos refriendo a las posiciones tanto mínima como máxima que son válidas para un cuadro de desplazamiento. El programa es capaz de definirlos mediante enteros, aunque por defecto ya posee los valores predeterminados 0 y 100 respectivamente.

TIPOS DE BARRAS

Las barras de desplazamiento se pueden dividir en dos clases: por un lado tenemos las barras de desplazamiento propiamente dichas, y por otro los denominados controles de barra. Normalmente los controles de Windows se comportan como ventanas hijas y separadas que aparecen asociadas a la ventana padre y que además, poseen la propiedad de ser totalmente programables, pudiéndose colocar y dimensionar según la necesidad del desarrollador. La barra de desplazamiento por su parte, tiene un tamaño y posición en la ventana predeterminados según sea el tamaño y ubicación del área cliente de la ventana en la que se utiliza.

FUNCIONES DE LAS BARRAS

Las funciones *GetScrollPos()* y *SetScrollPos()* se utilizan para poner u obtener el valor de la posición dentro de una barra de desplazamiento. Ambas envían o reciben un valor de 16 bits.

Existen dos clases de barras: los controles y las propias barras de desplazamiento

Además de las dos mencionadas, en Windows de 32 *bits* (95/98/NT) existen dos funciones adicionales: *GetScrollInfo()* y *SetScrollInfo()*. La primera utiliza tres parámetros: un *handle* de la ventana, el *flag fnBar* y un puntero a una estructura. El *flag fnBar* puede poseer tres valores: *SB_CTL* (para un control), *SB_HORZ* (barra de desplazamiento horizontal) y *SB_VERT* (barra de desplazamiento vertical). La estructura mencionada a la que apunta el tercer parámetro es la siguiente:

```
typedef struct tagSCROLLINFO
 {
  UINT cbSize
  UINT fMask;
  int nMin;
  int nMax;
  UINT nPage;
  int nPos;
  int nTrackPos;
 } SCROLLINFO, FAR *LPSCROLLINFO
typedef SCROLLINFO CONST FAR *LPSCRO-
        LLINFO;
```

Respecto a la estructura anterior, sólo debemos mencionar que *cbSize* se refiere al tamaño de la propia estructura, y *fMask* es una combinación de los valores: *SIF_ALL*, *SIF_DISABLENOSCROLL*, *SIF_PAGE*, *SIF_POS* y *SIF_RANGE*.

Por otro lado disponemos de la función *SetScrollInfo()* donde se requieren cuatro parámetros: los tres de la anterior más un valor de tipo *Boolean (TRUE* o *FALSE)*. Si se indica el primero de estos valores se dibuja el botón del cuadro de desplazamiento. En caso contrario dicho cuadro no hará su aparición.

EJEMPLO DE UN PROGRAMA

Dentro del listado 1 se ha incluido el código fuente de un programa C para Windows que hace uso de las barras de desplazamiento. De esta manera el lector podrá apreciar cómo se aplican todas las instrucciones relacionadas con esta materia en un ejemplo genérico de su uso.

Listado 1. Programa básico de ejemplo.

```
#include <windows.h>
#include <winuser.h>
#include <stdio.h>
#include <string.h>

LRESULT CALLBACK
WndProc(HWND, UINT, WPARAM,
LPARAM);
Char szProgName[]= "ProgName";

Int WINAPI WinMain(HINSTANCE
hInst, HINSTANCE hPreIns, LPSTR.
LpszCmdLine, int nCmdShow)
{
 HWND hWnd;
 MSG lpMsg;
 WNDCLASS wcApp;

 WcApp.lpszClassName  = szProgName;
 WcApp.hInstance      = hInst;
 WcApp.lpfnWndProc    = WndProc;
 WcApp.hCursor        =
LoadCursor(NULL, IDC_ARROW);
 WcApp.hIcon          = 0;
 WcApp.lpszMenuName = 0;
 WcApp.hbrBackground =
GetStockObject(LTGRAY_BRUSH);
 WcApp.style          = CS_HRE-
DRAW | CS_VREDRAW;
 WcApp.cbClsExtra     = 0;
 WcApp.cbWndExtra     = 0;
  If(!RegisterClass (&wcApp))
    Return 0;

 hWnd= CreateWindow(szProgName,
"Ejemplo 1. Un programa que usa barras
de desplazamiento", WS_OVERLAP-
PEDWINDOW, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDE-
FAULT, CW_USEDEFAULT, (HWND)
NULL,  (HMENU) NULL, (HANDLE)
hInst, (LPSTR) NULL);

 ShowWindow(hWnd, nCmDShow);
 UpdateWindow(hWnd);
 While (GetMessage(&lpMsg, NULL, 0,
0))
    {
      TranslateMessage(&lpMsg);
      DispatchMessage(&lpMsg);
    }
 return(lpMsg.wParam);
}

LRESULT CALLBACK
WndProc(HWND hWnd, UINT messg,
WPARAM wParam, LPARAM lParam)
 {
  HDC hdc;
  PAINTSTRUCT ps;
  TEXTMETRIC tm;
  char szbuffer[160] = " ";
  static int charwt, charht, HOldPos,
HNewPos, VOldPos, VNewPos, nDelta;
  FILE *fp;
  int i,t,ch;
  SCROLLINFO si;

  switch (messg)
   {
    case WM_CREATE:
    hdc= GetDC (hWnd);
    GetTextMetrics( hdc, &tm);
    charwt = tm.tmAveCharWidth;
    charht = tm.tmHeight +
tm.tmExternalLeading;
    ReleaseDC( hWnd, hdc);
    break;

    case WM_HSCROLL;
    si.cbSize = sizeof(si);
    si.fMask = SIF_ALL;
    si.nMin = 0;
    si.nMax = 100;
    si.npage = 10;
    si.nPos = HNewPos;
    SetScrollInfo(hWnd, SB_HORZ, &si,
FALSE);

    HOldPos = si.nPos;

    switch (LOWORD(wParam))
     {
      case SB_LINERIGHT:
       nDelta = 1;
       break;
      case SB_LINELEFT:
       nDelta = -1;
       break;
      case SB_PAGERIGHT:
       nDelta = 15;
       break;
      case SB_PAGELEFT:
       nDelta = -15;
       break;
      case SB_THUMBPOSITION
       nDelta = ( HIWORD(wParam)) -
HOldPos;
       break;
      default:
       nDelta = 0;
       break;
     }

   if(nDelta)
    {
     HNewPos = HOldPos + nDelta;
     if(HOldPos < si.nMin)
       {
       nDelta - = HOldPos - si.nMin;
       HNewPos = si.nMin;
       }
    if (HOldPos > si.Max)
      {
      nDelta = - HOldPos - si.nMax;
      HNewPos = si.nMax;
      }
    }
   if(nDelta)
    {
     SetScrollPos(hWnd, SB_HORZ,
HNewPos, TRUE);
     si.nPos = HNewPos;
     SetScrollInfo( hWnd, SB_HORZ;
&si, FALSE);
    }
   InvalidateRect(hWnd, NULL, TRUE);
   break;

   case WM_VSCROLL:
    si.cbSize = sizeof(si);
    si.fMask = SIF_ALL;
    si.nMin = 0;
    si.nMax = 100;
    si.nPage = 10;
    si.nPos = VNewPos;
    SetScrollInfo( hWnd, SB_VERT,
```

ANÁLISIS DEL EJEMPLO

Como se puede observar en el código fuente con el que estamos trabajando, este ejemplo necesita para funcionar correctamente un fichero de texto denominado **texto.doc** que el programa se encargará de abrir y cuyo contenido podrá ser visualizado mediante las barras de desplazamiento que previamente han sido programadas para la ventana.

El código correspondiente a la carga del fichero no supone ningún problema de entender ya que no incluye ninguno de los conceptos específicos de la programación bajo Windows y básicamente es código C típico del que cualquier programador medio ya ha usado en alguna ocasión.

Listado 1. Programa básico de ejemplo. (Continuación).

```
&si, FALSE);

    VOldPos = si.nPos;

    switch( LOWORD( wParam))
      {
      case SB_LINEDOWN:
        nDelta = 1;
        break;
      case SB_LINEUP:
        nDelta = -1;
        break;
      case SB_PAGEDOWN:
        nDelta = 15;
        break;
      case SB_PAGEUP:
        nDelta = -15;
        break;
      case SB_THUMBPOSITION:
        nDelta = (HIWORD( wParam))
- VOldPos;
        break ;
      default:
        nDelta = 0;
        break;
      }

    if(nDelta)
      {
      VNewPos = VOldPos + nDelta;
      if(VOlPos < si.nMin)
        {
        nDelta - = VOldPos - si.nMin;
        VNewPos = si.nMin;
        }
      if(VOlPos > si.nMax)
        {
        nDelta - = VOldPos - si.nMax;
        VNewPos = si.nMax;
        }
      }

    if(nDelta)
      {
      SetScrollPos( hWnd, SB_VERT,
VNewPos, TRUE);
      si.nPos = VNewPos;
      SetScrollInfo( hWnd, SB_VERT,
&si, FALSE);
      }
    InvalidateRect( hWnd, NULL,
TRUE);
    break;

    case WM_PAINT:
      hdc = BeginPaint ( hWnd, &ps);

      // Funciones CGI
      t = 0;
      if (( fp= fopen("c:/texto.doc", "r"))
!= NULL)
        {
        while(!feof(fp))
          {
          ch = fgetc(fp);
          i = 0;
          while((ch!='\n') && (ch!=
EOF))
            {
            szBuffer[i] = (char)ch;
            ch = fgetc(fp);
            i++;
            }
          textOut( hdc, -charwt
*(HNewPos),
                charht *(t -
VNewPos), szBuffer,i);
          t++;
          }
        }
      fclose( fp);

      // Fuciones de CGI antes:
      ValidateRect( hWnd, NULL);
      EndPaint( hWnd, &ps);
      break;

      case WM_DESTROY:
      PostQuitMessage(0);
      break;

      default:
        return( DefWindowProc( hWnd,
messg, wParam, lParam));
    }
    return(0);
  }
```

La apertura del fichero de texto se hace de forma normal, o sea, se indica el *path* u *filename* del mismo, de forma que se obtiene un *handle* al mismo. A partir de aquí podemos ya leer su contenido.

Un control normalmente aparece como una ventana hija dentro de una ventana padre

Para poder acceder a su contenido se ha creado un bucle bidimensional donde se leen los caracteres del fichero por líneas (al final de cada una de las cuales ésta se escribe en el búffer de pantalla) y dura infinitamente hasta que se lea el carácter de fin de fichero (OEF), momento en que el bucle se da por finalizado.

Realizada la lectura del fichero, se cierra.

Otro aspecto que se puede observar consiste en la aparición de una función nueva denominada *GetTextmetrix()* que no había sido explicada hasta ahora. Se utiliza para consultar y manejar todos los valores relacionados con los tipos de fuentes empleados por el programa, lo cual, debido a la extensión del tema, se tratará en el próximo artículo con mayor profundidad.

WM_HSCROLL y *WM_VSCROLL* se utilizan en el programa para que se puedan procesar los mensajes generados por Windows relacionados con las barras de desplazamiento. Como se puede comprobar, los dos segmentos de código que tratan las dos barras, tanto la horizontal como la vertical, son muy parecidos.

El código necesario para gestionar la barra horizontal y vertical es prácticamente el mismo

Una vez que se ejecuta el programa, si el usuario pulsa sobre una de las flechas de desplazamiento de la barra vertical, Windows mandará a la aplicación un mensaje *SB_LINEUP* o

Listado 2 Ejemplos.

1. Dibujar un rectángulo:

```
Rectangle(hdc, 25,300,130,375);
TextOut(hdc, 50, 325, "Rectángulo", 16);
```

2. Dibujar un círculo:

```
Ellipse(hdc, 375, 75, 525, 225);
TextOut(hdc, 435, 190, "Círculo", 16);
```

3. Dibujar una elipse:

```
Ellipse(hdc, 275, 300, 200, 250);
TextOut(hdc, 450,190, "Elipse",16);
```

4.Dibujar una línea diagonal y un texto:

```
MoveToEx(hdc, 20, 20 ,NULL);
LineTo(hdc, 80, 80);
TextOut (hdc, 50, 25, "<-línea diagonal",16);
```

SB_LINEDOWN. Por otro lado, si el usuario pulsa en algún punto de la barra que no sea el cuadro de desplazamiento, se generará un mensaje de la clase *SB_PAGEDOWN* o *SB_PAGEUP*, que el programa ejecutará como un desplazamiento de 15 líneas del texto ya sea hacia abajo o hacia arriba.

GetScrollPos() y SetScrollPos() se utilizan para poner u obtener el valor de la posición dentro de una barra de desplazamiento

En todos los casos de desplazamiento sobre alguna de las barras, el mensaje se interpreta, se cambia el valor de nDelta y se actualizan todas aquellas posiciones de los cuadros de desplazamiento.

MÁS PRUEBAS

Como se dijo al final del artículo anterior de esta serie, una vez se comprende el funcionamiento de cada parte del código fuente dado de ejemplo, el usuario puede empezar a practicar la inserción de líneas de código para practicar funciones gráficas. Se caracterizan por su sencillez de uso, aunque es necesario disponer de la información técnica sobre el número y función de los parámetros utilizados en cada una de ellas.

Finalizamos con algunos ejemplos con los que podreis practicar y que detallamos a lo largo del listado 2.

Concurso de Programación

Como recordaras teníamos preparado un concurso de programas de ajedrez, pues bien .... el mismo queda convocado para los dias 12 y 13 de Septiembre, siendo la fecha límite de recepción de programas el dia 9 del mismo.

Para más información e inscripciones, puedes ponerte en contacto con nosotros en el teléfono 91 559 82 86 y pregunta por la Srta. SANDRA.

SÓLO PROGRAMADORES

AWS

Programación del API de Windows (II)

Jordi Agost (agost@eup.udl.es)

En el presente artículo vamos a profundizar en el funcionamiento de las llamadas a la API, para adquirir una base teórica, sobre la cual nos apoyaremos posteriormente para la realización de los ejemplos prácticos.

En primer lugar veremos cómo pasar matrices a un procedimiento de una DLL, para después estudiar la forma de pasar los tipos definidos por el usuario. Además veremos cómo pasar punteros de función a procedimientos de una DLL, para finalmente abordar los errores más usuales en las llamadas a la API.

Si lo que deseamos es pasar un elemento individual de una determinada matriz, entonces lo que haremos será pasar el elemento como el mismo tipo de la matriz. Por ejemplo para reproducir un determinado sonido de una lista haríamos:

```
Dim Wav(5) As String
Dim i As Integer, i2 As Integer

For i = 0 to Ubound(Wav)
    i2 = sndPlaySound(Wav(i), 0)
Next i
```

Pero si lo que deseamos es pasar una matriz completa a un determinado procedimiento de una DLL, pasaremos el primer elemento de la matriz por referencia (esto funciona en matrices numéricas ya que en dicho tipo de matriz los elementos se almacenan secuencialmente en memoria), y a partir de ese momento el procedimiento de la DLL tendrá acceso a toda la matriz.

PASO DE TIPOS DEFINIDOS POR EL USUARIO A UN PROCEDIMIENTO DE UNA DLL

Algunos procedimientos de *DLLs* toman como argumentos tipos de datos definidos por el usuario (a través de la sentencia *Type ... End Type*), es lo que en Pascal conocemos como registros y en C/C++ como estructuras. Al igual que en el apartado anterior de las matrices, podemos pasar los elementos individuales de un tipo definido por el usuario de la misma manera que pasamos variables numéricas o de cadena normales. Para pasar un tipo que haya definido el usuario como un único argumento, lo deberemos hacer mediante referencia, sin olvidar que los tipos definidos por el usuario nunca pueden pasarse por valor. En este último caso, Visual Basic pasaría la dirección en memoria del primer elemento, y todos los demás elementos se almacenarían en memoria después de este primero. Vamos a ver paso a paso un ejemplo para explicar dichos conceptos.

En primer lugar creamos un nuevo proyecto, al que añadiremos un módulo de código como el siguiente:

```
Option Explicit
```

Demo disponible en nuestra Web.

Algunos Ratones son muy curiosos...

DadvinaFence
Seguridad en sus Aplicaciones.

Uno de los riesgos que sufren las aplicaciones informáticas es la posibilidad de su uso por personas no autorizadas.
DadvinaFence incorpora tres componentes OCX o VCL:**ApplicationLock, FormLock** y **NetMessenger**, proporcionando al programador un sistema de seguridad activo fácil de integrar y totalmente configurable.

- Versiones VCL para Delphi 3.0/2.0, C++ Builder 3 y OCX para Visual Basic 5.0.
- Asignación de permisos y privilegios sobre los objetos existentes en las pantallas de las aplicaciones.
- Autentificación de usuarios mediante el uso de identificación y contraseña.
- Detección de intentos fallidos continuados de entrada al sistema,avisando al instante a los administradores de seguridad de las aplicaciones.
- Centralización de los accesos (login) a las bases de datos de la aplicación.
- Registro de entradas y salidas.
- Sistema modular fácilmente actualizable en tiempo de ejecución y de diseño.

etc...

Sus clientes se lo Agradecerán

Sólo por 31.500* ptas.

Dadvina Systems.

San Toribio 6.
28031 Madrid.
mail: com@dadvina.com

Para más información o realizar pedidos contacte con el telefono 91 380 32 46 o en nuestra Web: www.dadvina.com.

* IVA no incluido.

```
Declare Function DrawFocusRect Lib "user32"
      (ByVal hdc As Long, lpRect As RECT) As
      Long
Declare Function InvertRect Lib "user32"
      (ByVal hdc As Long, lpRect As RECT) As
      Long

Type RECTAPI
  X1 As Long
  Y1 As Long
  X2 As Long
  Y2 As Long
End Type
```

La primera de las dos funciones que definimos sirve para dibujar un contorno de puntos alrededor del rectángulo especificado como argumento, y la segunda para invertir los colores de un rectángulo determinado. A continuación, declaramos un tipo definido por el usuario, el cual utilizaremos para definir nuestro rectángulo. Ahora, en la sección de declaraciones correspondiente al *form1* del proyecto, añadimos:

```
Option Explicit
Dim Rec As RECTAPI
```

Y en los respectivos procedimientos añadimos el siguiente código:

```
Private Sub Form_MouseDown(Button As Inte-
      ger, Shift As Integer, X As Single, Y As
      Single)
  ScaleMode = 3
  If Button And 1 Then
    Rec.X1 = X
    Rec.Y1 = Y
    Rec.X2 = X
    Rec.Y2 = Y
  End If
End Sub

Private Sub Form_MouseMove(Button As Inte-
      ger, Shift As Integer, X As Single, Y As
      Single)
  ScaleMode = 3
  If Button And 1 Then
    DrawFocusRect hdc, Rec
    Rec.X2 = X
    Rec.Y2 = Y
    DrawFocusRect hdc, Rec
  End If
End Sub

Private Sub Form_MouseUp(Button As Integer,
      Shift As Integer, X As Single, Y As Sin-
      gle)
  ScaleMode = 3
  If Not (Button And 1) Then
    Rec.X2 = X
    Rec.Y2 = Y
    InvertRect hdc, Rec
  End If
End Sub
```

Con la propiedad *ScaleMode* del formulario establecemos la unidad de medida de las coordenadas del formulario, y lo que hacemos en cada paso es ir dibujando el cuadrado o invirtiendo sus colores (cuando liberamos el botón) según convenga.

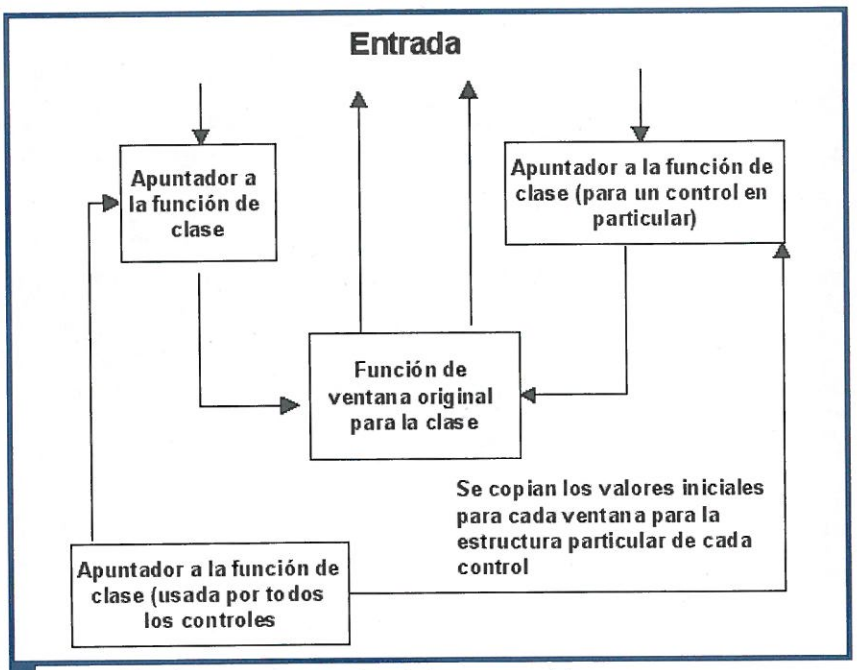

Entrada
Apuntador a la función de clase
Apuntador a la función de clase (para un control en particular)
Función de ventana original para la clase
Se copian los valores iniciales para cada ventana para la estructura particular de cada control
Apuntador a la función de clase (usada por todos los controles

Figura 1. Flujo de programa para una función de ventana estándar.

USO DE CONTROLADORES CON UN PROCEDIMIENTO DE UNA DLL

En el pasado artículo vimos que cuando pasábamos las propiedades de los objetos Visual Basic lo hacíamos por valor. Vamos a ver ahora cómo se pasan unas propiedades un poco especiales llamadas controladores. Un controlador es un valor de tipo *Long* definido por el sistema operativo con una longitud de 32 *bits*. Dicho objeto recibe también a veces, la denominación de "manipulador" (*handle*). Tiene la propiedad de ser único, y normalmente lo usamos para hacer referencia a objetos (dentro de Visual Basic estos objetos son normalmente los formularios o los controles). Muchos de los procedimientos de las *DLL* hacen usos intensivos de dichos controladores, ya sea en su forma de controladores de ventanas, *hWnd* (*handle to a Window*), o como controla-

Un controlador es un valor de tipo Long definido por el sistema operativo con una longitud de 32 bits

dores de dispositivos *hDC*, (*handle Device Context*) u otros.

Siempre que queramos pasar un controlador como argumento a un procedimiento de una *DLL* tendremos que declararlo como *ByVal Long*, y las funciones que a su vez devolverán un determinado controlador las declararemos de tipo *Long*. Si deseamos pasar algún controlador de alguno de nuestros objetos, usaremos la propiedad hWnd en el caso de los formularios o controles no gráficos y la propiedad hDC en el caso de tratarse de formularios y controles de cuadros con imágenes, y al igual que las propiedades pasaremos los controladores por valor y no por referencia.

La subclasificación permite interceptar los mensajes que Windows envía a un determinado formulario o control

Es necesario resaltar el hecho de que un controlador es simplemente un identificador, nunca será un puntero o un valor numérico, por lo tanto bajo ningún concepto debemos realizar ningún tipo de operación con él dentro de nuestro código.

APUNTADORES A FUNCIONES

Diremos que un puntero de función es una convención que permitirá pasar la dirección de una cierta función, la cual ha sido definida por el usuario como argumento a otra función, declarada para su uso con la aplicación. Muchas de las funciones de la *API* de Windows usan apuntadores a funciones como parámetros para permitir que Windows llame a una determinada función de una aplicación del usuario. Dicha función del usuario, la cual es llamada por Windows, se denomina **callback,** (función de retorno o función de devolución). Hasta la versión 5.0 de Visual Basic, este lenguaje no permitía este tipo de funciones, pero ahora podemos tener acceso a muchas funciones de la *API Win32* que anteriormente no eran admitidas desde Visual Basic. De todos modos nuestra libertad no es total a la hora de utilizar punteros de función, y tenemos algunas limitaciones como por ejemplo:

- **Argumentos incorrectos:** Si pasamos una función de retorno y llamamos por equivocación a un argumento con *ByRef* o *ByVal* posiblemente fallará la aplicación.
- **Punteros de Visual Basic a Visual Basic:** Dicha acción no es posible, actualmente sólo se admiten punteros desde Visual Basic a funciones de una librería *DLL*.
- **Depuración:** Si la aplicación activa una función de retorno mientras nos encontramos en modo de depuración, el código se ejecutará pero saltándose todas las interrupciones y los pasos que puedan llegar a existir.
- **Procedimientos inexistentes:** Al trabajar en modo de depuración puede ser que pasemos un puntero a una determinada función de Windows, pero cuando ya no existe la función subyacente (esto es muy probable que suceda con la función *WindowsProc*), provoca un fallo de protección general y posiblemente bloqueará todo el entorno integrado de Visual Basic.

Como nota final del apartado es necesario decir que cualquier código que utilicemos para llamar a un apuntador a una función deberá insertarse en un módulo estándar (un archivo de extensión *BAS*). Nunca deberemos colocarlo en un formulario ni en un módulo de clase.

SUBCLASIFICACIÓN

Conocemos como **subclasificación** a una técnica que permite interceptar los mensajes que Windows envía a un determinado formulario o control, y actuar en consecuencia, ya sea para ampliar la funcionalidad asociado al mismo o para anular su efecto. Se trata de una técnica bastante potente ya que, aparte de permitir la modificación de los controles estándar y los formularios, también permite acceder a mensajes internos de Windows que normalmente no se encuentran al alcance de los programadores de Visual Basic, como por ejemplo los eventos. Esta técnica se utiliza en el desarrollo de gran cantidad de controles de terceros, los cuales amplían las funcionalidades de los existentes.

Cualquier código que utilicemos deberá insertarse en un módulo estándar (un archivo de extensión .BAS)

Para entender la subclasificación deberemos fijarnos en que cada ventana de Windows (y recordemos que una ventana puede ser un formulario, un control, un icono del escritorio, el fondo del escritorio) posee una función Windows que procesa mensajes (al igual que Visual Basic procesa eventos). Además cada clase posee una función Windows que por defecto será la utilizada para cada ventana de

la clase, aunque es posible que una ventana no necesite usar la función establecida. Una ventana puede llamar a una determinada función de Windows definida por el usuario y dicha función establecida por el usuario puede llegar a procesar cualquier mensaje y acto seguido, si le interesa, puede volver a llamar a la función por defecto. De esta forma funciona la subclasificación.

En la figura 1 observamos el flujo de programa para una función estándar y en la 2 se aprecia el flujo de programa cuando en una función de ventana se ha usado la técnica de la subclasificación. En ambas, el flujo de programa que se recibe es el perteneciente a un mensaje que se envía a dicha ventana. Enviar un mensaje a una ventana es lo mismo que llamar a la función de ventana de una ventana. Además, la función que se quiere llamar se especifica por medio de la estructura de datos de la ventana para el control en particular. Declaramos en primer lugar en un módulo las constantes y declaraciones que vayamos a utilizar:

```
Declare Function CallWindowProc Lib "user32"
      Alias "CallWindowProcA" (ByVal
      lpPrevWndFunc As Long, ByVal hwnd As
      Long, ByVal Msg As Long,      ByVal
      Param As Long, ByVal lParam As Long)
      As Long

Declare Function SetWindowLong Lib "user32"
      Alias "SetWindowLongA" (ByVal hwnd
      As Long, ByVal nIndex As Long, ByVal
      dwNewLong As Long) As Long

Public Const GWL_WNDPROC = -4
Global lpPrevWndProc As Long
Global Hand As Long
```

A continuación, con las dos subrutinas siguientes vamos a hacer que nuestro programa intercepte el flujo de mensajes de Windows. En el procedimiento, **ProcMensaje** llamamos a la función *SetWindowLong* con el índice *GWL_WNDPROC* para crear una subclase de la clase de ventana utilizada para crear la ventana. A continuación, se utiliza la palabra clave *AddressOf* con una función de devolución de llamada (*WindowProc*) para interceptar los mensajes e imprimir sus valores en la ventana **Inmediato**:

```
Public Sub ProcMensaje()
lpPrevWndProc = SetWindowLong(Hand,
      GWL_WNDPROC, AddressOf Window-
      Proc)
End Sub
```

En el segundo procedimiento, llamado **ProcNormal**, desactivaremos la subclasificación reemplazando la devolución de llamada por el procedimiento original de Windows:

```
Public Sub ProcNormal()
  Dim temp As Long
  temp = SetWindowLong(Hand, GWL_WND-
      PROC, lpPrevWndProc)
End Sub
```

Y en el siguiente procedimiento imprimiremos en la ventana **Inmediato** los mensajes pertinentes:

Una ventana puede llamar a una determinada función de Windows definida por el usuario

```
Function WindowProc(ByVal hw As Long, ByVal
      uMsg As Long, ByVal wParam As Long,
      ByVal lParam As Long) As Long
      Debug.Print "Mensaje: "; hw, uMsg, wPa-
      ram, lParam
      WindowProc =
      CallWindowProc(lpPrevWndProc, hw,
      uMsg, wParam, lParam)
End Function
```

En el código de la carga del formulario se establece el valor inicial del *handle* de la ventana del formulario del siguiente modo:

```
Private Sub Form_Load()
      Hand = Me.hwnd
End Sub
```

Y en el código correspondiente a los botones, simplemente activamos o

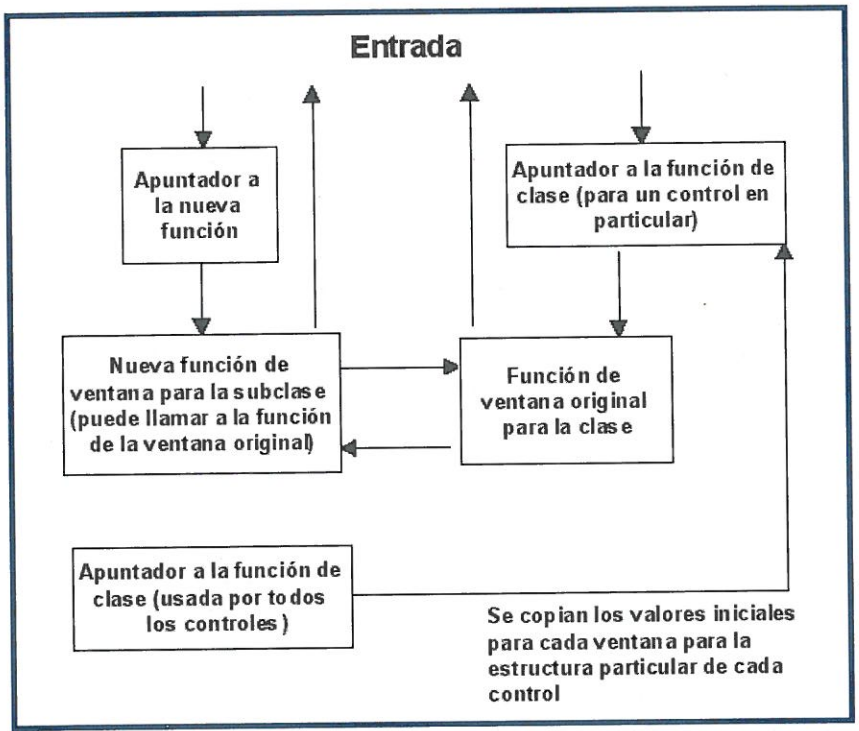

Entrada
Apuntador a la nueva función
Apuntador a la función de clase (para un control en particular)
Nueva función de ventana para la subclase (puede llamar a la función de la ventana original)
Función de ventana original para la clase
Apuntador a la función de clase (usada por todos los controles)
Se copian los valores iniciales para cada ventana para la estructura particular de cada control

Figura 2. Flujo de programa para una función de ventana subclasificada.

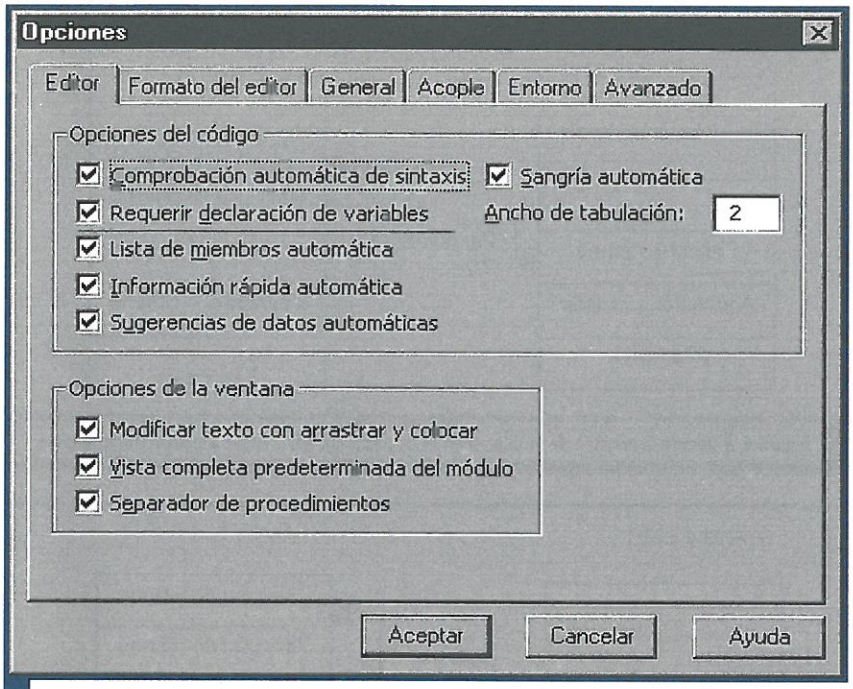


Figura 3. Hemos de marcar la declaración de variables.

desactivamos la subclasificación, según corresponda:

```
Private Sub Command1_Click()
    ProcMensaje
End Sub

Private Sub Command2_Click()
    ProcNormal
End Sub
```

API: GUÍA PARA EVITAR ERRORES

Visual Basic nos brinda un entorno relativamente seguro para desarrollar nuestros programas. A través de su lenguaje tenemos acceso de una forma encapsulada (es decir, no somos conscientes de ello) a una gran parte de las funcionalidades de la *API* de Windows. El precio que pagamos por ello consiste en encontrar ciertas limitaciones, con lo que a veces necesitaríamos hacer algo que no es posible. Con los procedimientos de las *DLLs* que acceden directamente a la *API* eliminamos dicha protección, aunque el precio que pagamos por ello es el aumento de la posibilidad de errores por nuestra parte.

A continuación vamos a especificar una serie de consejos o puntos que deberíamos revisar en caso de tener problemas con los accesos a la *API* de Windows.

Lo primero que debemos hacer al trabajar con Visual Basic es activar la opción **Requerir Declaración de variables** situada en el menú **Opciones** dentro del menú **Herramientas**, con el fin de obligar a que todas las variables que pasemos a una llamada a un procedimiento de una *DLL* estén declaradas. Será un primer filtro para evitar errores.

Después tendremos que **verificar los nombres de funciones.** Desde Visual Basic 4.0 los nombres de funciones de la *API* de Windows distinguen mayúsculas y minúsculas, además como explicamos en el artículo pasado, deberemos tener presente el uso de los sufijos *A* y *W* para funciones relacionadas con las cadenas. Para evitar este problema, lo más aconsejable es remitirnos al visor *API* (incluido como un **Add-In** del entorno de Visual Basic), el cual proporcionará la forma correcta de escribir cada declaración. Posteriormente deberemos comprobar **el uso correcto de la sentencia ByVal y el tipo de los parámetros,** siendo una de las causas más comunes a la hora de producir un error.

Si falta algún tipo de parámetro, Visual Basic proporcionará el error **Convención de llamada a DLL errónea** y si omitimos el tipo de un argumento que pasamos a Visual Basic, entonces este intentará pasar una variable de tipo *Variant* a una función *API* provocando así un error. Además, se debe tener en cuenta que, por ejemplo, si una declaración de la *API* espera una cadena pero recibe un valor numérico, Visual Basic convertirá dicho valor numérico en una cadena con resultados imprevisibles en nuestro programa.

Visual Basic nos da un entorno relativamente seguro para desarrollar nuestros programas

Otra fuente de error, que deberemos **comprobar es el tipo de retorno.** Si la función devuelve un resultado indicando si ha tenido éxito o no, deberemos chequear dicho valor para asegurarnos de que así sea. También deberemos **inicializar previamente las cadena con la longitud adecuada** antes de la llamada al procedimiento de la *API*. Deberemos recordar que las funciones *API* no tienen ningún procedimiento para saber si la longitud del área intermedia de variables es correcta, y todas las funciones presuponen que el programa que realiza la llamada se ha ocupado de reservar un espacio suficiente para las variables.

LAS LLAMADAS INTERNAS A FUNCIONES DE LA API

Ya que hemos dicho que el paso erróneo de parámetros así como la declaración errónea de sus tipos constituyen la forma más común de provocar un error, vamos a profundizar un poco más en las llamadas a la *API*. De esta forma tendremos la posibilidad de entender más a fondo nuestros errores y evitarlos.

Cada aplicación posee de forma interna una estructura de datos utilizada para el pase de parámetros y sus llamadas a funciones. Una pila además contiene también las variables locales de la función, asegurando así que la función tendrá su propio juego de variables locales, aún cuando ésta se llame a sí misma. De ahí el que sea de suma importancia la forma en que se colocan los parámetros de la pila. Así por ejemplo supongamos que tenemos una función hipotética que devuelve un tipo entero y pasa un argumento entero y uno de cadena:

```
Declare Function FuncionPrueba Lib
    "Prueba.dll" (Arg1 As Integer, Arg2 As
    String) As Integer
```

Cada aplicación posee de forma interna una estructura de datos utilizada para el pase de parámetros y sus llamadas a funciones

Si declarásemos la función erróneamente, por ejemplo si olvidamos el parámetro entero y hacemos una declaración del tipo siguiente:

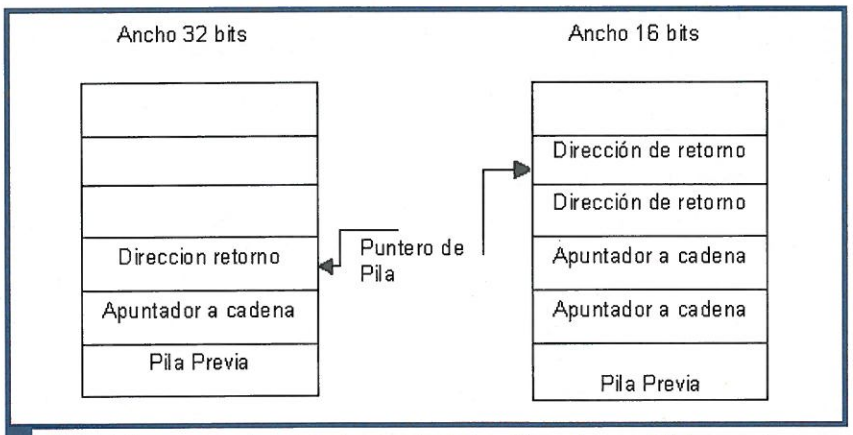


Figura 4. Estructura errónea de la pila, si en la declaración nos hemos olvidado un parámetro.

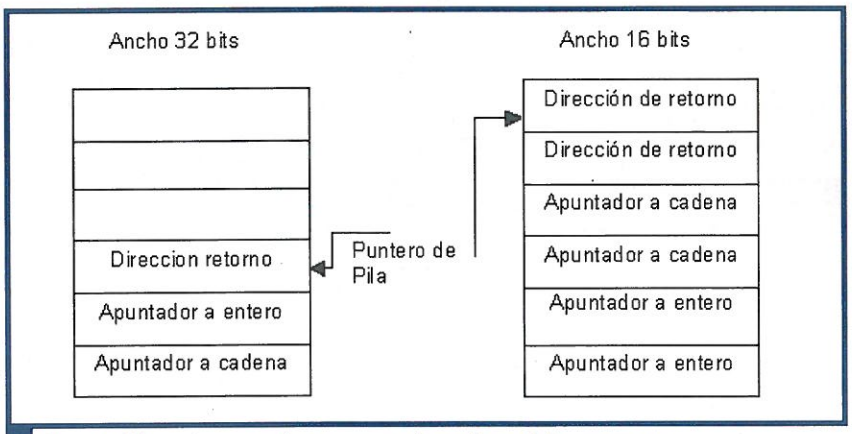


Figura 5. Pila errónea si nos hemos equivocado en el tipo de los argumentos.

```
Declare Function FuncionPrueba Lib
    "Prueba.dll" (Arg2 As String) As Integer
```

Veríamos como los datos se sitúan mal, tal y como indica la figura 4.

Aquí vemos como los argumentos se han colocado de una forma errónea. En la *API Win32* se interpretará como una cadena el contenido previo de la pila. Si la función grabase datos en la cadena se provocaría un error, ya que el valor en donde se escribiese debería ser una dirección de memoria. En *Win16*, debido a la diferente ordenación de los elementos de la pila, el parámetro sería correcto, pero el parámetro de tipo entero contendría basura.

Examinemos ahora la figura 5. En ella vemos reflejado el problema que se produce si nos equivocamos con el tipo de datos, y utilizamos el valor *long* en vez de utilizar el tipo *integer*. En *Win32* como cada variable toma como mínimo 32 *bits* (es decir, en una variable entera Visual Basic extendería el valor de 16 *bits* a una variable de 32 *bits*), la llamada al procedimiento funcionaría correctamente. Pero en *Win16* la llamada no encontraría la información correcta y el apuntador de la pila no estaría en el sitio correcto.

Hasta aquí, los conocimientos teóricos que son necesarios para empezar a trabajar con la *API* de Windows. En los artículos posteriores se aplicará la teoría explicada hasta el momento para ir desarrollando ejemplos prácticos de cómo usar las llamadas a la *API* de Windows, aumentando de esta forma la capacidad de nuestros programas.

Comparativa de herramientas Java

Ignacio Rojo Fraile (ignacior@jet.es)

El lenguaje Java se va perfilando progresivamente como el lenguaje de Internet. Un producto que cumpla la iniciativa 100% Pure Java nos asegura la compatibilidad multiplataforma, lo cual pone a Java en una posición privilegiada a la hora de desarrollar aplicaciones que incluso no interactúen con Internet.

A lo largo de este último año han ido apareciendo una serie de herramientas *RAD* (*Rapid Application Development*) cuyo propósito se basa en conseguir que la programación sea visual en la mayor medida posible y que el desarrollador tenga que teclear cuantas menos líneas de código mejor. A la hora de elegir una de estas herramientas *RAD* para Java tendremos que analizar distintos factores como que el código producido sea 100% Java Compatible, que ofrezca soporte para JavaBeans, que soporte acceso a bases de datos e incluso que tenga soporte *CORBA/RMI*. De entre todas las existentes en el mercado nos ha parecido interesante hacer un estudio de 5 herramientas: IBM VisualAge for Java 1.0 , Microsoft Visual J++ 6.0, Symantec Visual Café Database Development Edition 2.5, Borland Jbuilder 2 y Java Studio 1.0 de Sun.

IBM VISUALAGE FOR JAVA

VisualAge es una herramienta *RAD* que permite generar rápidamente *applets* Java y aplicaciones. Con el ratón se puede diseñar el interfaz del programa, definir el comportamiento (eventos) de los elementos de la interfaz, así como definir la relación entre la interfaz y el resto del programa. La propia herramienta se encarga de "convertir" la interfaz y sus correspondientes eventos diseñados con el ratón a código 100% compatible Java.

Cuenta con una serie de asistentes que entre otras cosas permiten crear nuevos *applets* o aplicaciones, crear componentes de aplicaciones (proyectos, paquetes, clases, interfaces y métodos), crear JavaBeans, importar y exportar código (ficheros **java** y **class**) desde y hacia el espacio donde trabajamos.

También incluye un almacén donde disponemos de las diferentes versiones de los programas. Otra cualidad ventajosa de esta herramienta reside en la posibilidad de modificar el código fuente durante la ejecución de un *applet* o aplicación: si queremos modificar el código de una aplicación que se está ejecutando, para observar el resultado del cambio no se necesita parar y reiniciar la ejecución.

En la figura 1 aparece un ejemplo de desarrollo de una interfaz y de la definición del comportamiento de sus componentes: al pulsar un elemento del campo *Listado de personal* le estamos indicando que lo copie al campo de texto *Nombre*.

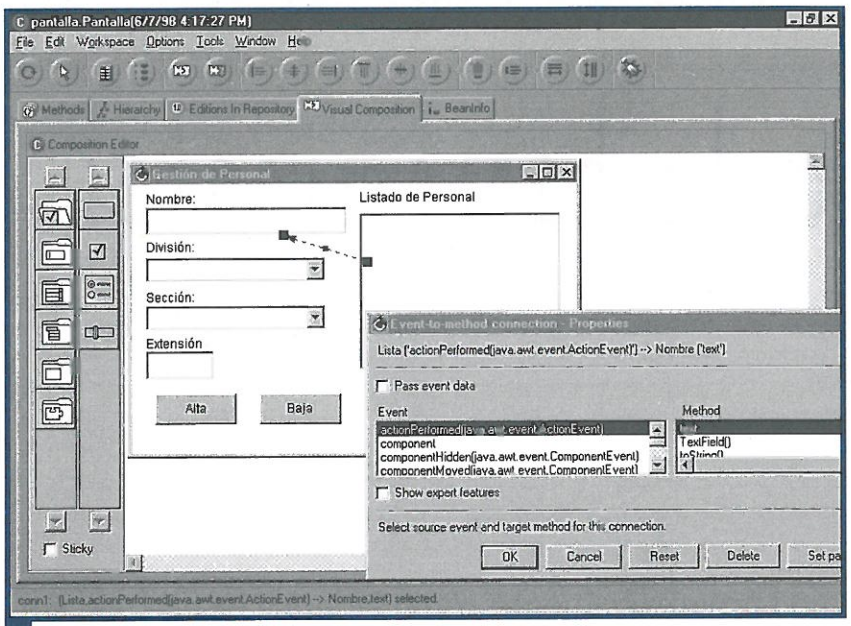


Figura 1. Generación de eventos mediante VisualAge.

Solamente con seleccionar la opción **Save Bean** del menú **File** esta herramienta se encarga de producir el código Java de la interfaz creada de forma visual utilizando el ratón. Los objetos son tratados como *Beans* pudiéndose editar sus propiedades mediante un formulario.

VisualAge permite tratar los *beans* creados por el usuario de igual manera que los *beans* que aporta la aplicación. Una vez creado un *bean*, se puede integrar dentro de la biblioteca de componentes a través del menú correspondiente.

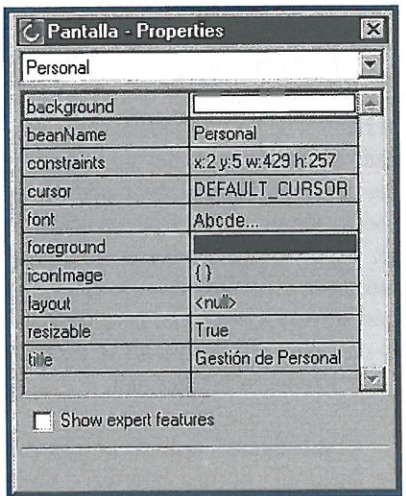


Figura 2. VisualAge permite tratar los beans creados por el usuario de igual manera que los beans que aporta la aplicación.

EL ESPACIO DE TRABAJO

Dentro del entorno de trabajo de VisualAge no se manipulan ficheros Java, sino que el código se trata en una base de datos de objetos estructurados que son mostrados al desarrollador como una jerarquía de componentes de programas.

> En el entorno de trabajo de VisualAge no se manipulan ficheros Java

Debido a que se manipulan componentes de programas en vez de ficheros, el desarrollo se centra en la organización lógica del código sin tener que preocuparse de los nombres de ficheros. Así podemos tener una ventana abierta con el código de una clase, de un método, constructor o una declaración de variables de manera independiente al resto de los componentes del fichero Java, lo cual aporta una gran claridad aún cuando los ficheros son de un tamaño considerable.

MANTENIMIENTO DE VERSIONES

VisualAge contiene un sofisticado sistema de gestión de código que posibilita el mantenimiento de múltiples ediciones de programas. Se puede establecer la versión de un proyecto, un paquete o una clase. Cuando se realiza esta operación en una edición de un programa en el repositorio, podemos manipular el código de la edición actual en el espacio de trabajo con la seguridad de que la edición a la damos una versión estará siempre disponible en dicho almacén.

> Permite mantener versiones de los diferentes componentes de cada programa

Toda actividad en VisualAge está organizada alrededor de un único espacio de trabajo que contiene el código de los programas sobre los que se trabaja. Permite mantener versiones de los diferentes componentes de cada programa. Podremos hacer comparaciones entre diferentes versiones y reemplazarlas. Este nivel de mantenimiento se puede efectuar tanto a nivel de proyecto, de paquete, de clase o de componente de una clase.

ACCESOS A BASES DE DATOS

La versión *Enterprise* proporciona conectividad con bases de datos a través de su herramienta propia: *Data Access Builder*. Esta herramienta permite que los *beans* accedan a datos mediante un asistente sobre tablas y

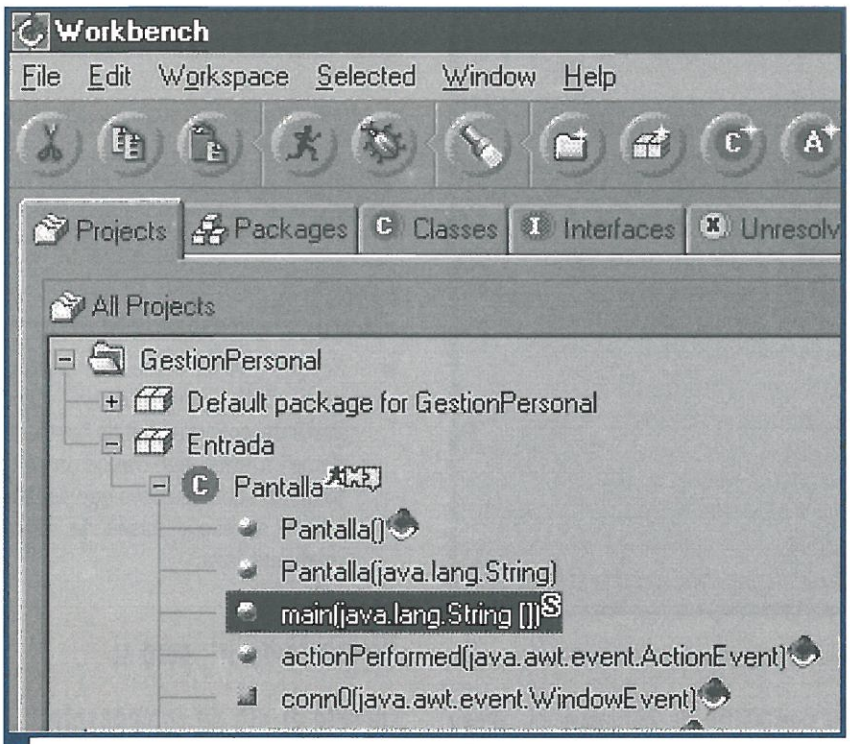


Figura 3. El código se trata en una base de datos de objetos estructurados que son mostrados al desarrollador como una jerarquía de componentes de programas.

campos. Mediante este asistente se pueden seleccionar los campos o bien escribir manualmente el código SQL.

CARENCIAS

- *Compatibilidad del código*: Aunque VisualAge genera código 100% compatible Java, no es capaz de soportar completamente el JDK 1.1. Por ejemplo, no se pueden programar clases internas con esta herramienta.
- *JavaBeans incluidos*: a excepción de los beans de acceso a bases de datos, la paleta no incluye más componentes que los estándar del *AWT*.
- *Aplicaciones distribuídas*: VisualAge *RMI Access Builder* crea las clases necesarias para crear aplicaciones *RMI* (*Remote Method Invocation*)pero no hay ninguna herramienta para *CORBA*.

CONCLUSIONES

El entorno de trabajo VisualAge resulta muy sencillo de manejar, a la vez de que es una de las herramientas más completas del mercado. Mantiene la compatibilidad Java 100%, con las carencias indicadas anteriormente. En cualquier caso ofrece muchas ventajas y escasos inconvenientes.

MICROSOFT, VISUAL J++ 6.0 PREVIEW

Visual J++ 6.0 es un producto que permite generar aplicaciones Java de una forma intuitiva y rápida. Partiendo del entorno Microsoft para otros lenguajes, el denominado *Visual Studio* utilizado por los programadores de Visual C++ o Visual Basic, nos encontramos con un cómodo y potente entorno de edición, con el compilador Java más rápido del mercado y con un depurador muy manejable y eficaz. También se debe destacar la disponibilidad de diversos asistentes que facilitarán la realización de las tareas más comunes. Vamos a fijarnos en las principales características de Visual J++ que son las siguientes, tal y como vamos a ir detallando paso a paso a continuación:

- *Compatibilidad del código*: Plena compatibilidad con Java (mientras no se use *ActiveX* ni los últimos añadidos del producto). Todo lo que se desarrolló con Java es compatible con Visual J++. Los archivos de clases que se generan pueden ejecutarse en cualquier otra plataforma.

Visual J++ 6.0 permite generar aplicaciones Java de una forma intuitiva y rápida

- *Potentes asistentes*: Entre los asistentes cabe destacar el VJ98 *Application Wizard* que permite generar cualquier tipo de aplicación de una forma cómoda, sin necesidad de escribir líneas de código.
- *Rapidez*: El compilador de Visual J++ es el más rápido del mercado (compila más de un millón de líneas por minuto, según Microsoft). También optimiza el código sin perder la compatibilidad con Java.
- *Depuración de errores*: Dispone de un depurador integrado en el propio entorno de trabajo que permite comprobar el estado de las variables, pila de llamadas, activar puntos de control, etc.

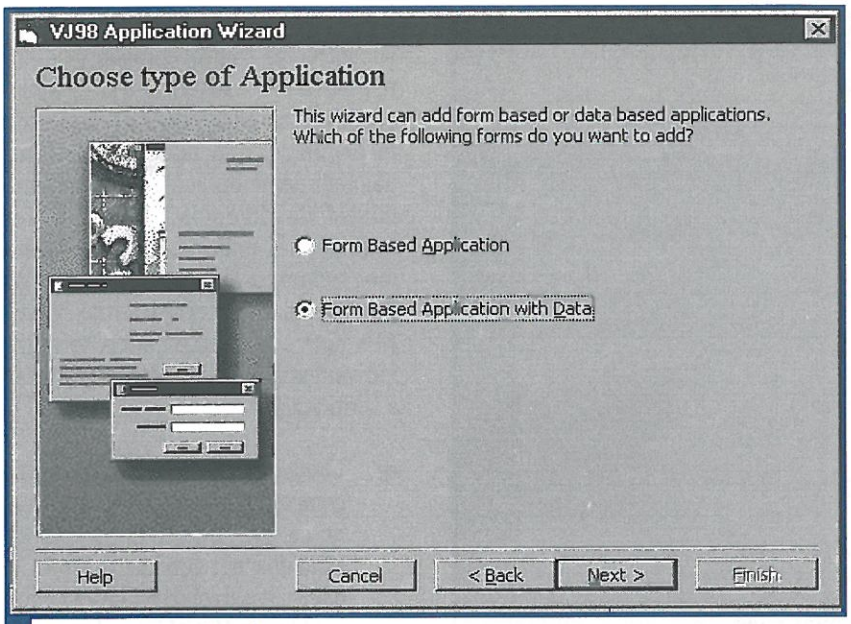

VJ98 Application Wizard
Choose type of Application
This wizard can add form based or data based applications. Which of the following forms do you want to add?
Form Based Application
Form Based Application with Data
Help
Cancel
< Back
Next >
Finish

Figura 4. Asistente de aplicaciones.

- *Compatibilidad de versiones*: Mantiene la compatibilidad con versiones anteriores como pueda ser Visual J++ 1.1.

- *Controles ActiveX*: A diferencia del producto anterior, en este aspecto destaca al incluir cientos de controles *ActiveX* para el desarrollo de potentes aplicaciones.

- *Generación de código*: Permite la generación de ficheros Windows EXE, COM DLL, CAB, ZIP, etc.

- *Bases de datos*: Esta herramienta ofrece la posibilidad de desarrollar aplicaciones que utilizan bases de datos mediante el uso de controles *ActiveX*.

Visual J++ 6.0 incluye entre sus características el nuevo concepto *"delegates"* en el lenguaje, fuera de las especificaciones Java de *Sun*. Se trata de una mejora en el tratamiento de punteros a funciones, muy requeridos en la mayoría de las aplicaciones. Visual J++, al igual que los sistemas operativos Windows son productos de la misma empresa, por lo que Microsoft ofrece con su nueva herramienta la posibilidad de optimizar el código Java, eso sí, a cambio de perder la compatibilidad del lenguaje.

Es un producto ideal para aquellos que se quieran iniciar en la programación con Java

Hay que hacer mención a un tema que siempre rodea a las herramientas Visual J++. Se trata del compilador *JIT* (*just-in-time*). En contra de lo que Microsoft anuncia y afirma, Visual J++ no consta de un *JIT* propio, sino que es una parte separada que funciona con el navegador Explorer.

Si bien este producto no consta de *JIT*, sí soporta el uso de componentes *ActiveX*, por lo que podemos importar código comercial (*controles ActiveX*), acceder a bases de datos mediante estos componentes, etc. Esto puede parecer un paso adelante pero más bien es un añadido que hace que Visual J++ pierda la compatibilidad Java justo en el momento en que usemos un componente *ActiveX*.

Otro importante elemento para la generación de aplicaciones gráficas es el asistente para aplicaciones, que nos permite generar aplicaciones sin apenas tener que escribir código Java, ya que gracias a este asistente se generarán las clases y los métodos necesarios.

CARENCIAS

- *Soporte JDBC*: Ausencia del soporte *JDBC* (*Java Database Connectivity*, conectividad de bases de datos en Java), que ofrece un conjunto de controladores que permiten el acceso a bases de datos mediante Java.

No consta de JIT, pero si soporta el uso de componentes ActiveX

- *Compatibilidad del código*: Es una versión 1.1 compatible Java, por lo que actualmente requiere que actualicemos el compilador mediante algunas técnicas algo complejas si deseamos un entorno 100% compatible Java.

CONCLUSIONES

Como se ha indicado en los puntos anteriores, Visual J++ 6.0 contiene todas las características necesarias para el desarrollo de aplicaciones de una manera simple y eficaz, por lo que es un producto ideal para aquellos que se quieran iniciar en la programación con Java. Cabe destacar que debido a que Microsoft ha decidido perder la "compatibilidad Java 100%", el uso de Visual J++ 6.0 en aplicaciones "portables" queda bastante restringido. Es decir, el aprovechamiento de todas las posibilidades ofrecidas por el producto redunda en la pérdida de compatibilidad con el lenguaje Java.

VISUAL CAFE 2.5 FOR DATABASE DEVELOPMENT EDITION

Herramienta RAD para la creación de *applets* y aplicaciones Java con soporte total para bases de datos. Ofrece un entorno de herramientas integrado entre las que encontramos un gestor de proyectos, diseñador de formularios, librería de componentes, editor de código fuente, editor de clases, asistente para la creación de eventos, asistente para bases de datos (tanto a través de *JDBC* como a través de arquitectura *thre-tier*), así como un depurador gráfico.

Permite observar, editar y modificar la jerarquía de clases utilizadas durante el desarrollo applets y aplicaciones

La creación de *applets* puede realizarse siguiendo una plantilla que actúa a modo de *esqueleto,* existiendo una diferente según el tipo de aplicación que se desee obtener (*applet* básico, *JavaBean*, aplicación de consola, aplicación GUI para Windows, etc.). Visual Café destaca por su facilidad de manejo (los asistentes facilitan el desarrollo de *applets*), además de que éstos ofrecen una disponibilidad total sobre el código fuente. Otro característica destacada consiste en la posibilidad de utilizar una versión para Macintosh.

En el paquete vienen integrados Visual Page (Editor HTML), *dbANYWHERE Server* (producto ajeno a Visual Café que alojándolo en el servidor permite que las aplicaciones y *applets* acceder a sus datos), *Sybase SQL Anywhere* (Base de datos relacional) y *Netscape Fastrack Server*.

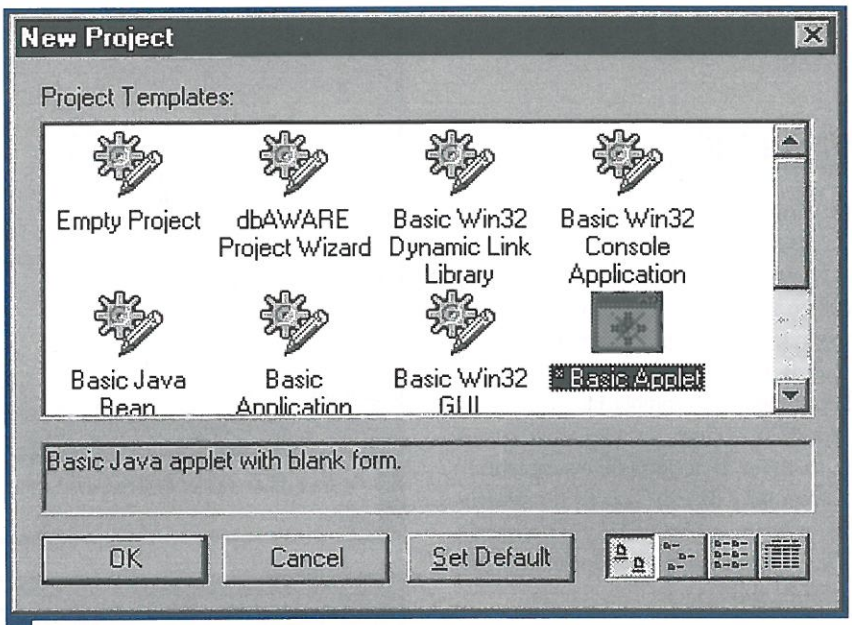


Figura 5. Visual Café permite crear applets siguiendo una plantilla que actúa a modo de "esqueleto".

EL INSPECTOR DE CLASES

Esta herramienta permite observar, editar y modificar la jerarquía de clases utilizadas durante el desarrollo de *applets* y aplicaciones. Mediante *Drag-&Drop* se puede modificar la jerarquía y obtener el resultado tanto gráficamente como en código.

EL NAVEGADOR DE JERARQUÍAS

Ofrece una lista con las clases de un proyecto y muestra todos los métodos y datos dentro de una determinada clase, de tal forma que cada uno de ellos se puede editar individualmente. Esto facilita la navegación dentro de las áreas que queramos editar (independientemente del tamaño del código fuente) y reduce las posibilidades de modificar accidentalmente código que se encuentre fuera del alcance del objeto seleccionado.

EL DEPURADOR GRÁFICO MULTINIVEL

Permite manipular datos, llamadas a funciones e hilos. De esta forma se puede observar y modificar el valor de una variable y continuar la depuración sin tener que reiniciar la aplicación. Además Visual Café permite depurar un *applet* o una aplicación que se esté ejecutando en otra máquina.

BASES DE DATOS

Visual Café incluye varios asistentes que permiten crear formularios sobre tablas de bases de datos existentes, así como definir consultas y modificaciones de las mismas. Su amplio soporte a bases de datos es una de sus puntos fuertes. El *dbNavigator* proporciona una visión jerárquica de la estructura de las bases de datos utilizadas. Permite obtener vistas de los servidores *dbANYWHERE* instalados y las bases de datos disponibles, así como la distribución de filas y columnas por cada una de las tablas. Con *dbNavigator* podemos añadir de forma sencilla campos de la base de

datos en el formulario de la aplicación o *applet* desarrollado.

CONCLUSIONES

Visual Café es una de las herramientas cuyo manejo resulta más fácil. Sus asistentes para la generación de nuevo código son excelentes, siendo ideal para programadores con conocimientos muy básicos que no necesiten dominar una herramienta más complicada. No por fácil deja de ser eficaz, ya que como hemos visto ofrece la mayoría de las características propias de este tipo de herramientas. Además, soporta *RMI* y serialización *IDL*, de tal forma que será posible la creación de aplicaciones distribuídas.

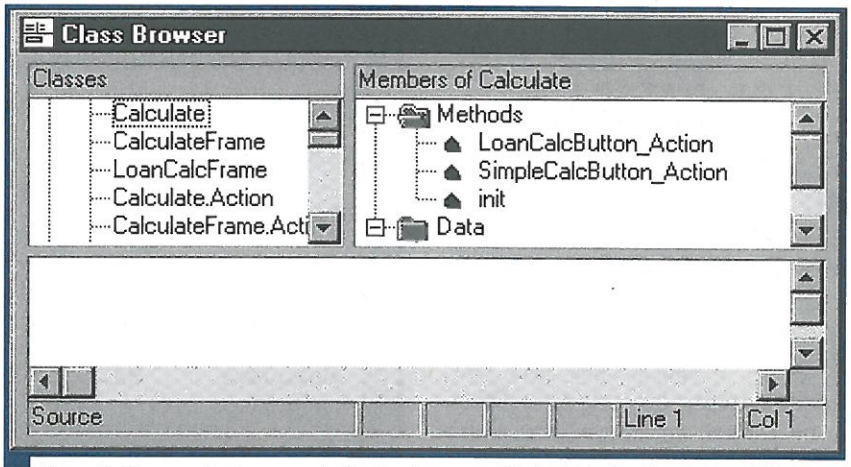


Figura 6. El navegador de jerarquías facilita la navegación dentro de las áreas que queramos editar.

Figura 7. El entorno de Jbuilder tiene bastante en común con otras herramientas como Delphi o Symantec Visual Café.

BORLAND JBUILDER 2

El entorno de JBuilder tiene bastante en común con otras herramientas como Delphi o Symantec Visual Café. Está formado por una barra de control flotante y una ventana de navegación denominada *AppBrowser*. La barra de control está compuesta por una serie de menús dispuestos en la parte superior, una paleta de iconos en la parte izquierda que proporcionan acceso directo a acciones de estos menús y una serie de componentes disponibles para usarlos en la aplicación o *applet*.

Está formado por una barra de control flotante y una ventana de navegación denominada AppBrowser

La ventana de navegación tiene dos árboles expansibles a la izquierda y una ventana de edición a la derecha. El árbol situado en el marco superior contiene la lista de ficheros del proyecto y el otro contiene un análisis jerárquico de la estructura del fichero seleccionado. Lo que se pretende es que haya una única ventana por proyecto, de tal forma que se podrán tener abiertos varios proyectos al mismo tiempo, cada cual dentro de su correspondiente ventana. El contenido de la ventana de navegación puede variar dependiendo de si se está editando/compilando, diseñando la interfaz o durante el proceso de depuración. Cada una de estas ventanas de navegación permite que pulsando las pestañas situadas en su parte inferior podamos seleccionar el tipo de información que se desee visualizar. Para observar la jerarquía de clases simplemente hay que seleccionar el fichero Java en el panel de navegación e ir desplegando el árbol.

JBuilder 2 se basa en el *JDK* de *Sun*, con algun pequeño cambio para permitir su adaptación a la herramienta. El código generado es 100% puro Java, ofreciendo la posibilidad de elegir que se ejecute y compile con cualquiera de las versiones existentes de JDK entre la 1.0, y 1.1.5. Soporta todas las funcionalidades del *JDK*: *JFC*, *JDBC*, *JNI*, *RMI*, Internacionalización, *RMI*, clases internas, *JavaBeans*, *JARs*, *IIOP* y *CORBA*. De nuevo se aprecia el esfuerzo de Borland (Inprise en la actualidad) de hacer sus herramientas compatibles con todas las nuevas tecnologías existentes en el mercado.

BeansExpress consiste en una serie de asistentes con elementos de desarrollo visual para la generación de JavaBeans

Incluye más de doscientos *JavaBeans* con su código fuente. Aporta una utilidad denominada *BeansEx-*

press que consiste en una serie de asistentes con elementos de desarrollo visual para la generación de *JavaBeans*. Permite tanto la modificación de componentes ya existentes como la transformación de una clase Java en un *JavaBean*. El formato del *bean* podrá ser *Zip*, *Zip* comprimido, *Jar* y *Jar* comprimido. Una vez que hayamos creado el componente será posible incorporarlo a la paleta de componentes.

Cuando se efectúan cambios visualmente no introduce etiquetas de código

JBuilder se encuentra dentro de las denominadas *herramientas de dos vías*. Si producimos una modificación en el código se ve reflejada en el diseño visual y viceversa. A diferencia de otras herramientas, cuando se efectúan cambios visualmente no introduce etiquetas en el código, por lo que no existe el peligro de que la aplicación deje de funcionar si eliminamos dichas etiquetas. El método para establecer propiedades a los objetos es muy similar al de VisualAge y Visual Café, con su correspondiente ventana para el trabajo con ellas. Además permite seleccionar varios objetos al mismo tiempo y modificar las propiedades comunes a todos ellos.

JBuilder incorpora diferentes asistentes para:

- *Creación de ficheros*: están disponibles en la galería de objetos. Permiten crear nuevos componentes partiendo de unos esqueletos predefinidos. Muy interesante entre ellos es el destinado a producir *servlets*. Produce la página HTML que llama al *servlet* y genera automáticamente la clase que contiene el *servlet* que trata los parámetros enviados.
- Utilidades de modificación de ficheros existentes disponibles en el menú *Wizards*.
- Asistente para objetos distribuidos.

Entre sus aportaciones destacan:

- Incluye componentes *JFC Swing* en la paleta de componentes. Además completa éstos con otros propios llamados componentes *dbSwing* que proporcionan acceso a bases de datos.
- *Borland Deployment Server*: Servidor de aplicaciones cuyo fin es que se pueda acceder a aplicaciones Java como si de *applets* se tratase, pero sin las restricciones de estos últimos. Las aplicaciones son almacenadas en el servidor y se puede acceder a ellas a través de clientes remotos bajo cualquier plataforma. Además asegura que todos los clientes estarán ejecutando la última versión de nuestra aplicación.

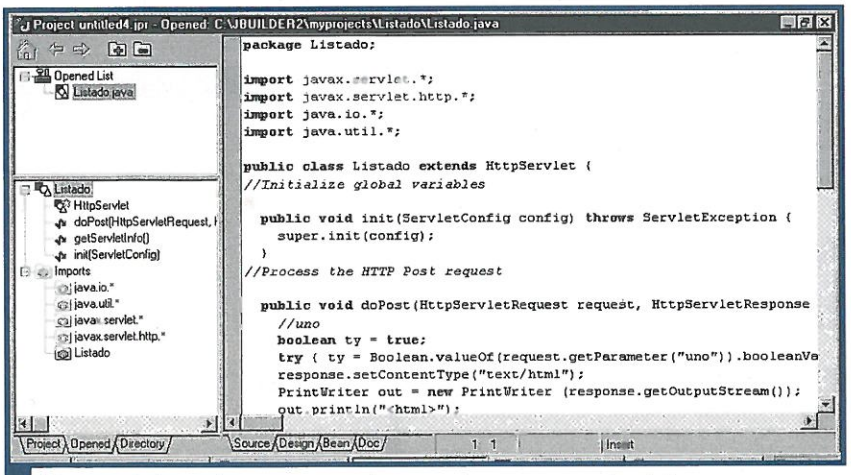


Figura 8. Cada una de estas ventanas de navegación permite que pulsando las pestañas en su parte inferior podamos seleccionar el tipo de información que queremos ver.

Borland Deployment Server permite acceder a aplicaciones Java como si de applets se tratase pero sin las restricciones que poseen éstos últimos

- *Aplicaciones distribuidas*: Soporta *VisiBroker 3.2*, lleva integrado el procesamiento de ficheros *IDL* y *RMI* e incluye un asistente para *CORBA*.
- La librería de componentes propia (*JBCL*)

CONECTIVIDAD A BASES DE DATOS

La versión cliente/servidor incorpora la arquitectura *DataExpress*. Los componentes *JBCL DataExpress* para acceso a bases de datos han sido desarrollados enteramente usando la *API* de *JDBC*. Además nos permite acceder a un servidor a través de un *driver ODBC*. Para ello se incluye en el paquete el software de conexión *JDBC-ODBC*. Esta versión incluye también otras algunas herramientas relacionadas con el acceso a bases de datos relacionales como pueden ser: *SQL Builder*, *SQL Monitor* y *SQL Explorer*.

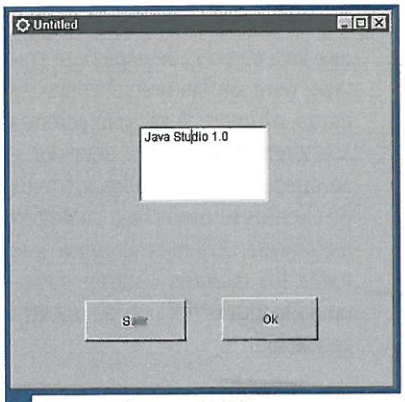


Figura 9. Ventana de GUI.

JBUILDER DATAGATEWAY

Proporciona a los desarrolladores Java una conectividad con bases de datos rápida, distribuida y que se adapta al estándar de *JDBC*. Consiste en una conexión pura Java al *BDE* (*Borland Database Engine*), que permite acceder a un amplio número de servidores *SQL* así como a pequeñas bases de datos en el propio terminal. *DataGateway* proporciona conectividad nativa a *Oracle*, *Sybase*, *DB2*, *Microsoft SQL Server*, *Informix*, *InterBase*, *Paradox*, *dBASE*, *Microsoft FoxPro* y *Microsoft Access*. Además ofrece además conectividad a través de *drivers* estándar *ODBC*. El *driver* cliente *JDBC* ha sido escrito completamente en Java y puede correr sobre cualquier máquina virtual Java que soporte JDK 1.1. El servidor *DataGateway* puede instalarse en Windows NT, Windows 95, y Windows 98.

CONCLUSIONES

Podemos asegurar que JBuilder 2 es la herramienta más completa. Su desarrollo visual entra dentro de las denominadas herramientas de dos vías. Sus facilidades para la conexión de bases de datos son muy completas y soporta completamente el JDK 1.1.5. y CORBA. Debemos destacar *BeansExpress* para el desarrollo de *Javabeans* y el gran número de *beans* que aporta.

SUN MICROSYSTEMS, JAVA STUDIO 1.0

Java Studio es un producto que permite el desarrollo de aplicaciones Java mediante el simple uso del ratón, gracias a un entorno muy intuitivo y práctico, de forma que el uso del teclado para la mayoría de las tareas es mínimo. *Sun* destaca que con su producto se pueden generar aplicaciones sencillas sin saber nada del propio lenguaje, lo cual en la mayoría de las ocasiones es totalmente cierto.

Mediante Java Studio se pueden crear *applets* que incorporen una gran variedad de componentes predefinidos que nos encontramos en el entorno de desarrollo. También permite el desarrollo de *JavaBeans* de una forma sencilla, así como realizar una importación/exportación.

CARACTERÍSTICAS

- Un entorno de desarrollo sencillo, práctico e intuitivo, de fácil aprendizaje al no contener demasiados componentes ni excesivas opciones. Todo lo que se visualiza es utilizable la mayoría de las veces, lo que demuestra el carácter práctico del producto.

Java Studio es capaz de crear applets que incorporen una gran variedad de componentes predefinidos

- Nos permite un desarrollo sencillo de aplicaciones, *applets* y *JavaBeans* mediante unos sencillos pasos, que van desde seleccionar los componentes, indicar sus características, relacionarlos entre sí, determinar las condiciones y probar.
- Mantiene la compatibilidad Java 100% con todos los componentes y todas las aplicaciones desarrolladas, de forma que no tendremos los clásicos problemas de que faltan unos paquetes por incluir ni similares.

Mantiene la compatibilidad Java 100% con todos los componentes y aplicaciones

- Consta de un tutorial elemental, una ayuda sencilla y completa que permite un aprendizaje rápido del producto.

El desarrollo de una aplicación pasa por un diseño de la misma en una ventana denominada *GUI: The user Interface* (*Graphics Users Interface*: Interfaz de usuario gráfica). En esta ventana iremos situando a nuestra elección los componentes que encontramos en las "pestañas", como son Multimedia, Internet, Database, etc.

Siguiendo con el diseño de nuestra interfaz utilizando botones, listas,

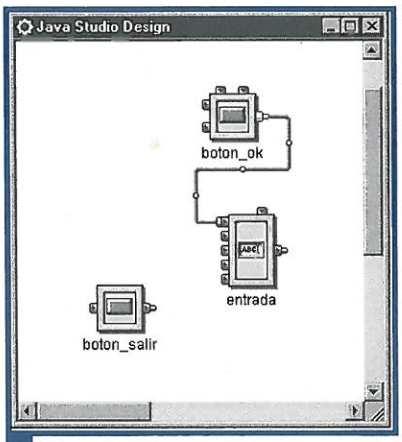


Figura 10. Ventana Java Studio Design.

Tabla 1. Comparativa de características de las herramientas.

	Visual Age 1.0	Visual J++ 6.0	Symantec Visual Café 2.5	Java Studio 1.0	Borland JBuilder 2.0
Disco duro necesario(MB)	100	31	35	40	165
Soporte JDK 1.1	●	●	●	●	●
RAD	●	●	●	●	●
JavaBeans	●	○	●	●	●
Gestor de Clases	●	●	●	○	●
RMI/CORBA	○	○	○	○	●
Soporte JDBC	●	○	●	●	●
Control de versiones	●	○	●	○	●
Creación EXE	○	●	●	○	○
Depurador Integrado	●	●	●	●	●
Precio (ptas.)	303.000	(1)	93.980	16.400	(2)
Nota final	**8**	**7**	**7**	**6**	**9**

Sí: ● No: ○

(1) Precio no disponible al tratarse de una preview.
(2) Existen dos precios, el standard de 15.000 pesetas, y el Cliente/Servidor de 365.000 pesetas.

mensajes, continuaremos determinando sus características, como el tamaño, evento al que responden, código que se ejecuta, etc. Una vez determinadas las características pasaremos a establecer las acciones entre los componentes, las condiciones a cumplir para ciertas ejecuciones, etc.

Necesita tener instalada una versión JDK para su funcionamiento, lo que complica la instalación inicial y el desarrollo posterior

Todas estas relaciones quedan definidas en otra ventana, que aparentemente se asemeja a un mecano por la forma de representación de los componentes unidos con "tuberías". Se trata de la ventana *Java Studio Design*, en la que se definen las relaciones, condiciones y acciones entre los componentes que estamos utilizando. En la figura 10 se observan la ventana de *Java Studio Design* con un ejemplo:

Una vez finalizada la definición de las acciones, relaciones, etc., probaremos la aplicación desarrollado y por último generamos el código resultante mediante la opción **Generate**. De esta forma podremos crear un *applet*, un *JavaBean* o una aplicación de la forma más simple.

CARENCIAS

- Necesita tener instalada una versión de *JDK* previamente para su funcionamiento, lo que complica la instalación inicial y el desarrollo de los primeros pasos, principalmente para los que se quieren iniciar en este entorno.
- Requiere de un ordenador potente y con mucha memoria, lo que hace a veces lento el desarrollo cuando se generan aplicaciones voluminosas.

Java Studio requiere un ordenador potente y con mucha memoria que a veces hace lento el desarrollo de las aplicaciones voluminosas

CONCLUSIONES

Usando Java Studio podemos dejar patente la facilidad con la que se pueden desarrollar aplicaciones Java, sin ser un experto en la materia ni conocer el lenguaje, por lo que podemos asegurar desde el principio, que se trata de una herramienta ideal para escribir los *applet* y aplicaciones personales, de una forma sencilla y rápida, manteniendo la compatibilidad Java a todos los niveles.

VALORACIÓN FINAL

Elegir una herramienta pasa por la naturaleza de la aplicación que se va a desarrollar a la hora de trabajar. Para aplicaciones profesionales la elección pasa por JBuilder o VisualAge. Si se desea crear un *applet* que dote de interaactividad una página *HTML* y se dispone de unos conocimientos mínimos, recomendamos la herramienta Java Studio.

Visual Café es una herramienta que está orientado principalmente a otro tipo de aplicaciones que no sean tan sencillas y que contengan bases de datos, pero sin llegar a la consideración de grandes proyectos.

Visual J++ da un aspecto y una sensación muy similar a lo que supone su producto hermano para C++ de Microsoft, centrándose de lleno en el código fuente.

> Seleccionar cualquier tipo de herramienta pasa en principio por la naturaleza de la aplicación que se va a desarrollar

VisualAge for Java es el encarga do de transformar el código fuente en su propio formato interno. Este estilo de representación estructurada de la información está el considerado más adecuado para el tratamiento de la misma.

JBuilder se puede considerar como la solución óptima para desarrollo cliente/servidor. Podemos catalogarla como la más aconsejable dentro de su grupo para el desarrollo de aplicaciones que por su envergadura requieran una herramienta de gran potencia.

Hasta aquí hemos llegado con la explicación de estas cinco herramientas, ahora sólo queda vuestra elección personal.

Ruiz Nicoli

Porque su futuro ya está cambiando. Porque están aprovechando sus propios recursos. Porque, trabajando juntos, estamos contribuyendo para que dejen de ser el Tercer Mundo.

Desde 1981

Ayuda en Acción

Sembrando Futuro

Integración de CGI en el servidor web (y V)

Enrique de la Lastra (elastra@redestb.es)

La interfaz WinCGI permite ampliar la funcionalidad de un servidor web, gracias a que logra el diálogo con aplicaciones especiales que proporcionan funciones adicionales a las propias de éste. Dichas funciones se concretan en la generación de páginas HTML dinámicas.

Si se desea consultar en un servidor web de una agencia de viajes sobre información relacionada con unas vacaciones en Canarias para un número de personas en unas fechas concretas y en un hotel con un determinado precio máximo, está claro que el servidor no puede disponer de una página HTML ya realizada con esos datos. Como la información solicitada por cada cliente puede tener muchos márgenes de variación, es ridículo pensar que existen en el servidor todas las páginas HTML que se adapten a todas las necesidades posibles de cada cliente.

La única manera de conseguir un resultado razonable para una consulta de este tipo es mediante la generación de páginas HTML dinámicas. Es decir, páginas HTML que no existen físicamente en el servidor, sino que se generan de forma dinámica, haciendo corresponder su contenido al resultado de una consulta a una base de datos donde se encuentra toda la información disponible.

Como quiera que un servidor web sólo entiende peticiones *HTTP* y sólo devuelve páginas HTML (y los archivos en ellas referenciados), se han ideado varios mecanismos para conseguir que llegue al servidor la información de una consulta personalizada de un cliente (en el sentido más amplio de la palabra) y que, a su vez, el servidor ejecute una aplicación que, con los datos de esa consulta, pregunte a una base de datos y genere una página HTML con la información pedida.

Los posibles medios de diálogo entre un servidor web y las aplicaciones que realizan funciones de consulta y generación de páginas HTML dinámicas son, entre otros: el *CGI* (*Common Gateway Interface*), el *WinCGI* (*Windows Common Gateway Interface,* esto es, el *CGI* para máquinas "Wintel"), las interfaces de *Servidor ISAPI* y NSAPI (*Internet y Netscape Server Application Programming Interface,* o interfaces de programación de aplicaciones de *Servidor de Microsoft* y de *Netscape*, respectivamente), las páginas *ASP* (*Active Server Pages*, Páginas de Servidor Activas de Microsoft) y muchas más.

En los dos anteriores artículos de esta serie vimos cómo crear un programa *WinCGI* y comprobamos su correcto funcionamiento con un servidor web comercial que cumple esta especificación: el *WebSite*. En este artículo, ampliaremos la capacidad del servidor web que creamos en los dos primeros artículos, para que cumpla la especificación *WinCGI* y de esta forma pueda enviar datos a aplicaciones *WinCGI* y devolver al cliente las páginas dinámicas generadas por aquéllas.

Hay mecanismos para enviar al servidor la consulta de un cliente y que aquél ejecute una aplicación que pregunte a una base de datos y genere una página HTML dinámica

¿QUÉ NECESITAMOS?

Lo primero y más importante que debemos tener en cuenta, es que hay que modificar el código necesario del servidor web, para que dé soporte al comando **POST**, ya que mediante este comando se pueden enviar datos de consulta o actualización al servidor web.

El segundo punto que tener en cuenta se centra en la definición de la interfaz con las aplicaciones de servidor. En los dos artículos anteriores entramos a conocer, más o menos en detalle, la especificación *WinCGI*. Una de las partes más importantes de esta especificación es la descripción de los posibles campos y valores de las distintas secciones del fichero de intercambio con extensión *ini*. Para que la aplicación *WinCGI* disponga de la máxima información posible, intentaremos rellenar dicho fichero con el mayor número de campos que podamos. Para ello, vamos a declarar una serie de variables que recogerán, de la propia petición *HTTP*, los valores de esos campos. A estas variables, las hemos procurado denominar igual que a los nombres de los campos del archivo *ini:* **RutaProgramaCGI** (*ExecutablePath*), **ParametrosCGI** (*QueryString*), *Referer, RequestMethod, RequestProtocol, RemoteAddress, RemoteHost,* Host (*ServerName*), *ServerPort, ServerSoftware, UserAgent, Connection.*

Habrá que definir también el nombre del fichero *ini* y el del fichero en el que el programa *WinCGI* generará la salida. Los nombres elegidos han sido, respectivamente, **CGI_SPWS.ini** y **CGI_SPWS.out** (las siglas *SPWS* corresponden a Sólo Programadores Web Server). Estos ficheros se generarán en un directorio del servidor donde crearemos y almacenaremos los archivos temporales. Este directorio lo denominaremos **/temp** y colgará directamente del raíz del servidor (**/home**).

Por último, será necesario definir una serie de variables y procedimientos nuevos, pero los iremos viendo a medida que vayan haciendo falta.

DISTINGUIR EL TIPO DE PETICIÓN

Cabe recordar primero que una petición **GET** a un programa *WinCGI*, no tiene por objeto "bajarse" el programa, sino ejecutarlo. Una petición **POST**, por su parte, además de llamarlo, le pasa cierta información de filtrado de parte del cliente. Por tanto, para distinguir si se trata de una petición de una página HTML o una llamada a un programa ejecutable (ya sea mediante el comando *GET* o mediante el comando *POST*), no utilizaremos el nombre del método de petición sino la ruta del recurso solicitado.

Si en la ruta, aparece el directorio **CGI_WIN**, suponemos que el recurso solicitado es en realidad un programa *WinCGI*. Para controlar este extremo, creamos la variable booleana **EsPeticionDePagina**. Si el cliente hace referencia a una página HTML estática, esta variable valdrá **True**, mientras que si la referencia es a un programa *WinCGI* tomará el valor **False**.

Debemos distinguir si la petición HTTP del cliente hace referencia a una página HTML o a un programa WinCGI

Dependiendo además del recurso solicitado, así será el valor del nombre del fichero físico en el servidor. En concreto, si la petición se dirige a un programa *WinCGI*, el fichero físico solicitado será el archivo de salida de la aplicación *WinCGI*:

```
NombreFicheroSolicitado := RecursoSolicitado;

if (RecursoSolicitado = '/') then begin
  EsPeticionDePagina := True;
  NombreFicheroSolicitado := DirectorioInicial
      + PaginaInicial;
end
else if (Pos('CGI-WIN', UpperCase(RecursoSoli-
      citado)) = 0) then begin
  EsPeticionDePagina := True;
  NombreFicheroSolicitado := DirectorioInicial
      + NombreFicheroSolicitado
end
else begin
  EsPeticionDePagina := False;
  NombreFicheroSolicitado := DirectorioInicial
      + '\temp\CGI_SPWS.out';
end;
```

PROCESAR LA PETICIÓN WINCGI

Para el caso en el que la petición del cliente (es decir, la *URL* introducida) haga referencia a un programa *WinCGI*, realizaremos un procesamiento del contenido de la petición

HTTP (en el procedimiento **ProcesarPeticionCGI**) para obtener los parámetros de la misma. Sabremos que la petición se refiere a una página o un programa *WinCGI* si la variable **EsPeticionDePagina** vale **False**. La lógica para generar el código *HTTP* de respuesta es la misma que en el caso de una petición *HTTP* convencional:

```
if (EsPeticionDePagina) then begin
...
end {if}
else begin {
 if (RequestMethod = 'GET') or (RequestMethod = 'POST') then begin
  ProcesarPeticionCGI (Socket);
  if FileExists(RutaProgramaCGI) then
   CodHTTP := 200
  else
   CodHTTP := 404;
 end
 else
  CodHTTP := 400;
end;
```

Sólo si el código *HTTP* de respuesta es **"200, OK"**, llamaremos al programa *WinCGI*. Para cumplir con la especificación, primero generaremos el archivo *ini* (**CGI_SPWS.INI**) que se pasará como parámetro en la llamada al programa *WinCGI*. Esto lo realizaremos en el procedimiento **CrearArchivoIniCGI**. Después, mediante la función **EjecutarProgramaCGI**, ejecutaremos la aplicación *WinCGI* correspondiente, pasándole como parámetro el archivo *ini*. Y si el proceso es satisfactorio, enviaremos al cliente el resultado de su petición (que en realidad habrá sido generado por el programa *WinCGI*). Esta última parte se desarrollará en el procedimiento **EnviarResultadoCGI**.

En caso de que la petición vaya dirigida a una página HTML o el código no sea **"200, OK"**, enviaremos las cabeceras y el recurso pedido, si corresponde:

```
if (not EsPeticionDePagina) and (CodHTTP = 200) then begin
 CrearArchivoIniCGI (Socket);
 if EjecutarProgramaCGI (Socket) then
  EnviarResultadoCGI (Socket);
end
else begin
 EnviarCabecerasHTTP (Socket);
 if (Peticion.RequestMethod = 'GET') and (CodigoHTTP =200) then begin
  AbrirFichero;
  EnviarFichero (Socket);
 end
 else begin
  Socket.Close;
 end;
end; {else}
```

Como hemos comentado, en el procedimiento **ProcesarPeticionCGI**, analizamos la petición *HTTP*, para obtener los parámetros de la consulta al programa *WinCGI*. El código completo se puede observar en el Listado 1; aquí sólo nos detendremos en los aspectos más importantes.

El servidor debe usar la función WinExec() para arrancar el programa CGI

La función **DevuelvePrimeraPalabra**, que se usa profusamente en este procedimiento, es una función de utilidad general, que toma como parámetros un *string* y un carácter "separador" y devuelve la parte del *string* que se encuentra antes del separador, o todo el *string* si el separador no aparece en la cadena de caracteres. A su vez deja en el parámetro *string* la parte posterior al separador, o una cadena vacía si no encuentra el separador. Esta función nos servirá, por tanto, para aislar las distintas cabeceras de la petición y para eliminar cualquier carácter que no sea útil.

Del recurso solicitado, obtenemos el nombre del programa *CGI*, con la ruta completa, y los parámetros de la consulta (enviados a través del comando *POST*, generalmente mediante un formulario HTML). Para aislar cada una de las partes, tenemos en cuenta cuál es el contenido de la petición HTTP que va a llegar al servidor. Ésta tendrá una forma semejante a cualquiera de las siguientes:

```
POST /cgi-win/cgi_01.exe HTTP/1.0
ó
POST /cgi-win/cgi_01.exe?articulo=ServidorWeb1.htm HTTP/1.0
ó
POST /cgi-win/cgi_01.exe/articulo/ServidorWeb HTTP/1.0
```

En la primera de las tres, no hay parámetros ó éstos se envían al final de la petición (que es lo habitual). En la segunda y la tercera, los parámetros van a continuación de la *URL*. En función de estos datos y mediante la función **DevuelvePrimeraPalabra** obtenemos el nombre del programa y los parámetros de la petición (ver Listado 1).

Para recoger las cabeceras de la petición, analizamos línea por línea el contenido de ésta. Como cada línea de una petición *HTTP* termina en los caracteres **#13#10** (es decir, un retorno de carro y un salto de línea), buscando el carácter *#10* obtenemos la línea completa hasta el retorno de carro. Cada línea, como ya sabemos, consta del nombre de una cabecera, seguido de dos puntos y del valor de la misma. Esto nos permite ir separando las distintas cabeceras e ir obteniendo, dentro de éstas, el nombre y el valor de cada una, como se puede ver en el código siguiente:

```
Cabecera := DevuelvePrimeraPalabra (Recibido, #10);
while (Recibido <> '') do begin
 Cabecera := DevuelvePrimeraPalabra (Recibido, ':');
 DevuelvePrimeraPalabra (Recibido, ' ');
 ValorCabecera := DevuelvePrimeraPalabra (Recibido, #13);
 if Cabecera = 'If-Modified-Since' then
  IfModifiedSince := ValorCabecera
 else if Cabecera = 'Referer' then
  Referer := ValorCabecera
 else if Cabecera = 'Connection' then
  Connection := ValorCabecera
```

suscríbete

a Sólo Programadores

y consigue un magnífico descuento

suscripción normal

ahorro **20%**

12 revistas (1 año) por sólo...

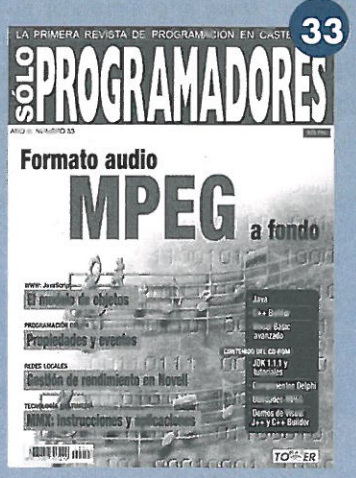
9.350 ptas.

suscripción estudiantes
(carreras técnicas)

ahorro **40%**

12 revistas (1 año) por sólo...

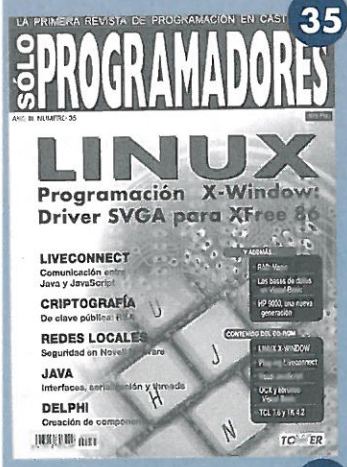
7.050 ptas.

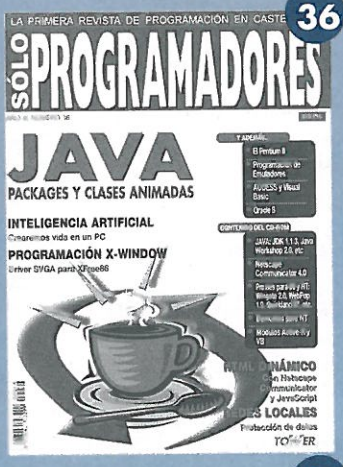


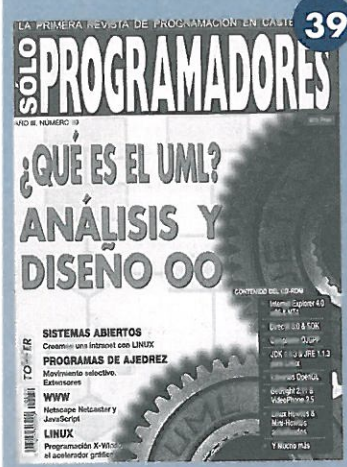
números atrasados

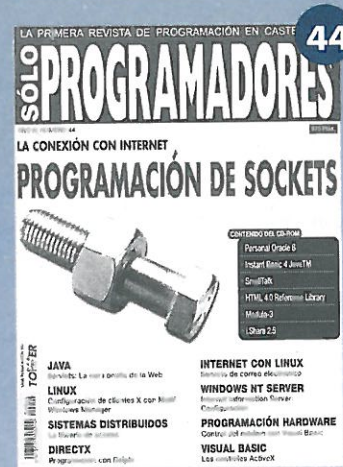
SÓLO PROGRAMADORES

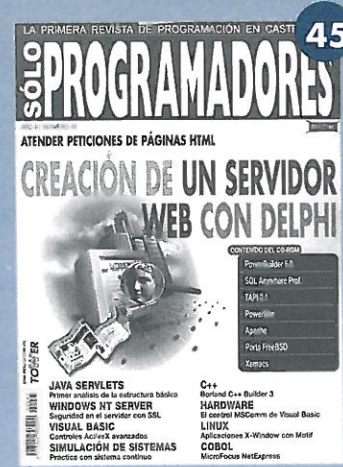
completa ya tu colección

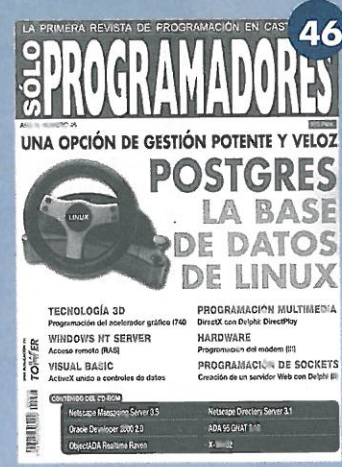












Listado 1. Código añadido o modificado en el servidor web para implementar la interfaz WinCGI.

```
procedure TForm1.WebServerSocket-
ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
var
  i: Integer;
begin
  Recibido := Socket.ReceiveText;
  if (Recibido = '') then exit;
  MemoRecibido.Lines.Add ('Cliente
Web: [' + Socket.RemoteAddress + ';' +
IntToStr(Socket.RemotePort) + ']');
  MemoRecibido.Lines.Add (Recibido);

  Peticion.RequestMethod := Devuelve-
PrimeraPalabra (Recibido, ' ');
  if (Pos('HTTP/1.0', Recibido) > 0) or
(Pos('HTTP/1.1', Recibido) > 0) then
begin
    RecursoSolicitado := DevuelvePri-
meraPalabra (Recibido, ' ');
    if (Pos('HTTP/1.0', Recibido) > 0)
then Peticion.RequestProtocol :=
'HTTP/1.0';
    if (Pos('HTTP/1.1', Recibido) > 0)
then Peticion.RequestProtocol :=
'HTTP/1.1';
  end
  else begin
    CerrarFichero;
    Socket.Close;
    exit;
  end;

  NombreFicheroSolicitado := Recurso-
Solicitado;

  For i:= 1 to Length(NombreFicheroSoli-
citado) do
    if (NombreFicheroSolicitado[i] = '/')
then NombreFicheroSolicitado[i] := '\';

  if (NombreFicheroSolicitado = '\') then
begin
    EsPeticionDePagina := True;
    NombreFicheroSolicitado := Directo-
rioInicial + PaginaInicial;
  end
  {Si no se pide la página por defecto,
comprobamos si se pide una página o se
hace una llamada a un programa CGI}
  else if (Pos('CGI-WIN',
UpperCase(RecursoSolicitado)) = 0) then
begin
    EsPeticionDePagina := True;
    NombreFicheroSolicitado := Directo-
rioInicial + NombreFicheroSolicitado
  end
  else begin
    EsPeticionDePagina := False;
    NombreFicheroSolicitado := Directo-
rioInicial + '\temp\CGI_SPWS.out';
  end ;

  {Si el comando es GET o HEAD y el
fichero existe, asignamos al código
HTTP el número 200 -OK-}
  if (EsPeticionDePagina) then begin
    if (Peticion.RequestMethod = 'GET')
or (Peticion.RequestMethod = 'HEAD')
then begin
      if FileExists(NombreFicheroSolicita-
do) then begin
        CodigoHTTP := 200
      end
      else
        CodigoHTTP := 404;
    end
    {si el comando no es GET ni HEAD
ni POST, como se ha pedido una página,
el código devuelto será 400 -Petición
errónea-}
    else
      CodigoHTTP := 400;
  end {if}
  else begin {-> not EsPeticionDePagina}
    if (Peticion.RequestMethod = 'GET')
or (Peticion.RequestMethod = 'POST')
then begin

      ProcesarPeticionCGI (Socket);
      if FileExists(Peticion.RutaPrograma-
CGI) then
        CodigoHTTP := 200
      else
        CodigoHTTP := 404;
    end
    {si el comando no es ni GET, ni
POST, el código será "400, Petición erró-
nea"}
    else
      CodigoHTTP := 400;
  end; {else}

  if (not EsPeticionDePagina) and (Codi-
goHTTP = 200) then begin
    CrearArchivoIniCGI (Socket);
    if EjecutarProgramaCGI (Socket) then
      EnviarResultadoCGI (Socket);
  end {if}
  else begin
    {Sea el comando que sea, enviamos
las cabeceras de la respuesta HTTP}
    EnviarCabecerasHTTP (Socket);
    {Si el comando es GET y el fichero
existe, abrimos el fichero -asignándolo al
Handle del fichero-, y lo enviamos }
    if (Peticion.RequestMethod = 'GET')
and (CodigoHTTP = 200) then begin
      AbrirFichero;
      EnviarFichero (Socket);
    end
    else begin
      Socket.Close;
    end;
  end; {else}
end;

procedure TForm1.ProcesarPeticionCGI
(Socket: TCustomWinSocket);
var
  i: integer;
  Cabecera, ValorCabecera: string;
begin
  ParametrosCGI := RecursoSolicitado;
  For i:= 1 to Length(ParametrosCGI) do
    if (ParametrosCGI[i] = '/') then Para-
metrosCGI[i] := '\';

  {Si recogemos dos veces la palabra
antes de '\', y dejamos en "RutaPrograma-
maCGI" el resto de la cadena, obtenemos
en RutaProgramaCGI una cadena con el
nombre del programa CGI y los paráme-
tros de la petición}
  RutaProgramaCGI := DirectorioInicial
+ DevuelvePrimeraPalabra (Parame-
trosCGI, '\');
  RutaProgramaCGI := RutaProgramaC-
GI + '\' + DevuelvePrimeraPalabra
(ParametrosCGI, '\');

  {Lo que haya antes del primer carácter
'?' ó '\' será el programa CGI y el resto,
los parámetros}
  i:= 1;
  while (ParametrosCGI[i] <> '?') and
        (ParametrosCGI[i] <> '\') and
        (i < Length(ParametrosCGI)) do
Inc(i);

  {Si el primer carácter después de la ruta
del programa CGI es un '?' o un '\'
extraemos los parámetros y la ruta del
programa. Si no es ninguno de los dos
significa que los parámetros se añaden al
final de la petición en lugar de seguir al
nombre del programa CGI}
  if ParametrosCGI[i] = '?' then
    RutaProgramaCGI := RutaProgramaC-
GI + '\' + DevuelvePrimeraPalabra
```

Listado 1. Código añadido o modificado al servidor web para implementar la interfaz WinCGI. (Continuación).

```
(ParametrosCGI, '?')
  else if ParametrosCGI[i] = '\' then
    RutaProgramaCGI := RutaProgramaC-
GI + '\' + DevuelvePrimeraPalabra
(ParametrosCGI, '\')
  else
    RutaProgramaCGI := RutaProgramaC-
GI + '\' + DevuelvePrimeraPalabra
(ParametrosCGI, ' ');

  {Recogemos las cabeceras, buscando
los saltos de línea, ya que cada cabecera
termina con un retorno de carro y un
salto de línea}
  Cabecera := DevuelvePrimeraPalabra
(Recibido, #10);
  while (Recibido <> '') do begin
    {El nombre de la cabecera irá antes de
los dos puntos. Quitamos el espacio pos-
terior a los dos puntos y obtenemos el
valor de la cabecera, que será lo que esté
situado antes del retorno de carro}
    Cabecera := DevuelvePrimeraPalabra
(Recibido, ':');
    DevuelvePrimeraPalabra (Recibido, '
');
    ValorCabecera := DevuelvePrimeraPa-
labra (Recibido, #13);      {Comparamos
el nombre de la cabecera con las cabece-
ras que vamos a tratar, y si coinciden,
asignamos el valor a la variable corres-
pondiente}
    if Cabecera = 'If-Modified-Since' then
      IfModifiedSince := ValorCabecera
    else if Cabecera = 'Referer' then
      Referer := ValorCabecera
    else if Cabecera = 'Connection' then
      Connection := ValorCabecera
    else if Cabecera = 'User-Agent' then
      UserAgent := ValorCabecera
    else if Cabecera = 'Pragma' then
      Pragma := ValorCabecera
    else if Cabecera = 'Host' then
      Host := ValorCabecera
    else if Cabecera = 'Accept' then
      Accept := ValorCabecera;

    {Eliminamos el salto de línea y busca-
mos de nuevo}
    DevuelvePrimeraPalabra (Recibido,
#10);
  end;

  if ParametrosCGI = '' then begin
    {Si los parámetros están al final de la
petición, la variable "ParametrosCGI"
estará vacía y los parámetros habrán que-
dado almacenados en la variable "Cabe-
cera"}
    DevuelvePrimeraPalabra (Cabecera,
#10);
    ParametrosCGI := DevuelvePrimera-
Palabra (Cabecera, #13);
  end;
end;

procedure TForm1.CrearArchivoIniCGI
(Socket: TCustomWinSocket);
var
  i: integer;
  FicheroIniCGI: TIniFile;
  Parametro, ValorParametro: string;
  Cadena: string;
begin
  NombreFicheroIniCGI := DirectorioIni-
cial + '\temp\CGI_SPWS.ini';
  FicheroIniCGI := TIniFile.Create
(NombreFicheroIniCGI);

  FicheroIniCGI.WriteString('CGI', 'CGI
Version', 'CGI/1.2 (Win)');
  FicheroIniCGI.WriteString('CGI', 'Exe-
cutable Path', RutaProgramaCGI);
  FicheroIniCGI.WriteString('CGI',
'Query String', ParametrosCGI);
  FicheroIniCGI.WriteString('CGI',
'Referer', Referer);
  FicheroIniCGI.WriteString('CGI',
'Request Method', RequestMethod);
  FicheroIniCGI.WriteString('CGI',
'Request Protocol', RequestProtocol);
  FicheroIniCGI.WriteString('CGI',
'Remote Address', Socket.RemoteAd-
dress);
  FicheroIniCGI.WriteString('CGI',
'Remote Host', Socket.RemoteHost);
  FicheroIniCGI.WriteString('CGI', 'Ser-
ver Name', Host);
  FicheroIniCGI.WriteString('CGI', 'Ser-
ver Port', IntToStr(Socket.LocalPort));
  FicheroIniCGI.WriteString('CGI', 'Ser-
ver Software', 'Servidor Web Solo Pro-
gramadores v0.2beta');

  Cadena := Accept;
  while (Cadena <> '') do begin
    ValorParametro := DevuelvePrimera-
Palabra (Cadena, ',');
    DevuelvePrimeraPalabra (Cadena, ' ');
    FicheroIniCGI.WriteString('Accept',
ValorParametro, 'yes');
  end;

  FicheroIniCGI.WriteString('System',
'GMT Offset', '-3600');
  FicheroIniCGI.WriteString('System',
'Debug Mode', 'No');
  FicheroIniCGI.WriteString('System',
'Output File', DirectorioInicial +
'\temp\CGI_SPWS.out');

  FicheroIniCGI.WriteString('Extra Hea-
ders', 'User Agent', UserAgent);
  FicheroIniCGI.WriteString('Extra Hea-
ders', 'Connection', Connection);
  FicheroIniCGI.WriteString('Extra Hea-
ders', 'Host', Host);

  if ParametrosCGI <> '' then
    Cadena := ParametrosCGI;

  For i:=1 to Length(Cadena) do
    if (Cadena[i] = '+') then Cadena[i] := '
';

  while (Cadena <> '') do begin
    ValorParametro := DevuelvePrimera-
Palabra (Cadena, '&');
    Parametro := DevuelvePrimeraPalabra
(ValorParametro, '=');
    FicheroIniCGI.WriteString('Form
Literal', Parametro, ValorParametro);
  end;

  FicheroIniCGI.Free;
end;

function TForm1.EjecutarProgramaCGI
(Socket: TCustomWinSocket): boolean;
var
  ProgramaCGI: array [0..255] of Char;
  StrAux : string;
  Ejecutado: boolean;
begin
  StrAux := RutaProgramaCGI + ' ' +
NombreFicheroIniCGI;
  StrPCopy(ProgramaCGI, StrAux);

  Ejecutado := True;
  try
    if WinExec(ProgramaCGI,
SW_SHOWNORMAL) < 32 then begin
      MessageDlg('No pudo ejecutarse '+
RutaProgramaCGI + '.', mtError,
[mbOk], 0);
      Ejecutado := False;
    end;
  except
    on EInOutError do begin
      MessageDlg('Directorio Inválido ('+
RutaProgramaCGI
```

Listado 1. Código añadido o modificado al servidor web para implementar la interfaz WinCGI. (Continuación).

```
+').',mtError,[mbOk],0);
      Ejecutado := False;
    end;
  end;

  if Not Ejecutado then begin
    BorrarFichero;
    Socket.Close;
    SysUtils.DeleteFile (NombreFicheroIniCGI);
  end;
  Result := Ejecutado;
end;

procedure TForm1.EnviarResultadoCGI (Socket: TCustomWinSocket);
begin
  AbrirFichero;
  EnviarFichero (Socket);
  BorrarFichero;
  Socket.Close;
  SysUtils.DeleteFile (NombreFicheroIniCGI);
end;

procedure TForm1.AbrirFichero;
begin
  if FileExists (NombreFicheroSolicitado) then begin
    AssignFile (HandleFicheroSolicitado, NombreFicheroSolicitado);
    Reset (HandleFicheroSolicitado);
    FicheroAbierto := True;
  end;
end;

procedure TForm1.CerrarFichero;
begin
  if FicheroAbierto then begin
    CloseFile (HandleFicheroSolicitado);
    FicheroAbierto := False;
  end;
end;

procedure TForm1.BorrarFichero;
begin
  CerrarFichero;
  SysUtils.DeleteFile (NombreFicheroSolicitado);
end;
```

```
  else if Cabecera = 'User-Agent' then
    UserAgent := ValorCabecera
  else if Cabecera = 'Pragma' then
    Pragma := ValorCabecera
  else if Cabecera = 'Host' then
    Host := ValorCabecera
  else if Cabecera = 'Accept' then
    Accept := ValorCabecera;
  DevuelvePrimeraPalabra (Recibido, #10);
end;
```

El resto del código está comentado en el propio Listado 1.

ARRANCAR EL PROGRAMA WINCGI

El servidor debe usar la función **WinExec()** de la *API* de Windows, para arrancar el programa *CGI*. La razón de recurrir a la *API* es que Delphi no dispone de ninguna función directa que permita ejecutar un programa externo. La función *WinExec()* toma dos parámetros: el nombre del ejecutable (en nuestro caso, el programa *WinCGI*) y el modo de presentación de la ventana (**SW_SHOWNORMAL**, es decir una ventana normal, sin maximizar ni minimizar). Si además queremos pasar algún parámetro al ejecutable, deberá ir después del nombre del programa, separado con un espacio en blanco. Si la función *WinExec()* tiene éxito, devolverá un valor entero mayor que **31**. Por tanto para llamar al programa *WinCGI* el código sería el siguiente:

```
var
  ProgramaCGI: array [0..255] of Char;
  StrAux : string;
begin
  StrAux := RutaProgramaCGI + ' ' + NombreFicheroIniCGI;
  StrPCopy(ProgramaCGI, StrAux);

  if WinExec(ProgramaCGI, SW_SHOWNORMAL) < 32 then begin
    MessageDlg('No pudo ejecutarse '+ RutaProgramaCGI + '.', mtError, [mbOk], 0);
  end;
```

Estas líneas, si las protegemos con una sentencia "try" quedan como se muestra en el Listado 1.

El fichero *ini*, debemos haberlo creado previamente y haberlo rellenado con los datos de la petición *HTTP*. En nuestro código, todo esto lo realiza el procedimiento **CrearArchivoIniCGI**. Mediante este procedimiento, creamos un archivo *ini* (**C:\home\temp\CGI_SPWS.ini**) y lo completamos con la información necesaria, siguiendo siempre la especificación *WinCGI*.

Una vez finalizado el proceso, cerramos el archivo y liberamos la memoria:

```
  NombreFicheroIniCGI := DirectorioInicial + '\temp\CGI_SPWS.ini';
  FicheroIniCGI := TIniFile.Create (NombreFicheroIniCGI);

  FicheroIniCGI.WriteString('CGI', 'CGI Version', 'CGI/1.2 (Win)');
  FicheroIniCGI.WriteString('CGI', 'Executable Path', RutaProgramaCGI);
...
FicheroIniCGI.WriteString('System', 'Output File', DirectorioInicial + '\temp\CGI_SPWS.out');
...
  FicheroIniCGI.Free;
```

Como el programa *WinCGI* termina inmediatamente después de realizar su función (gracias a la función *Halt* que irá al final del código, tal y como vimos en el anterior artículo), no hará falta mantener una sincronización con dicho programa, desde el servidor web, para detectar cuándo termina aquél. Por último, el procedi-

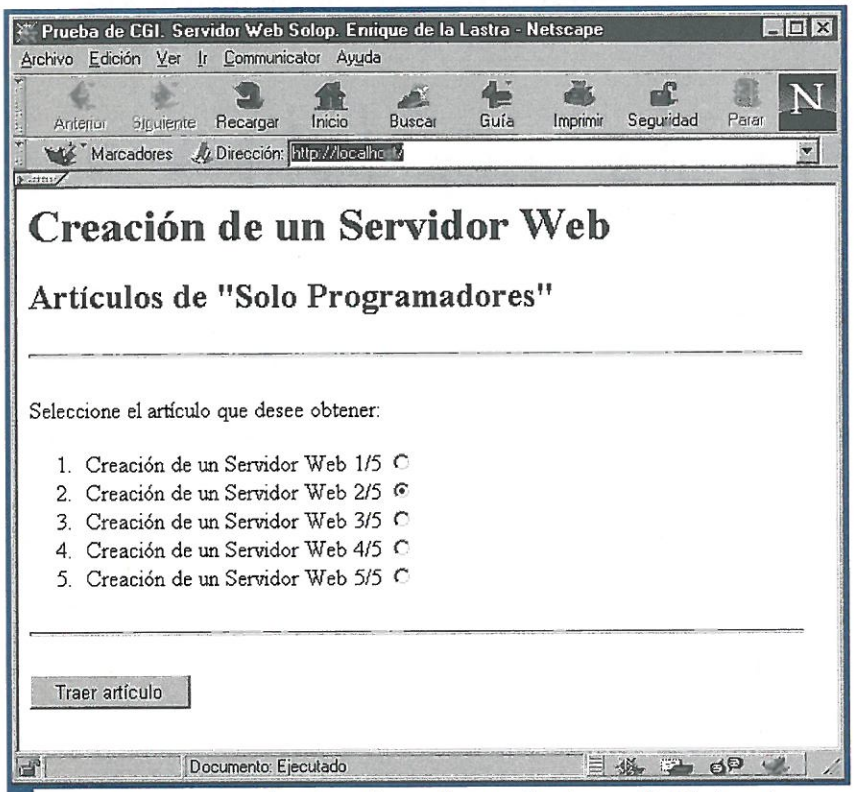

Prueba de CGI. Servidor Web Solop. Enrique de la Lastra - Netscape
Archivo Edición Ver Ir Communicator Ayuda
Anterior Siguiente Recargar Inicio Buscar Guía Imprimir Seguridad Parar
Marcadores Dirección: http://localhc
Creación de un Servidor Web
Artículos de "Solo Programadores"
Seleccione el artículo que desee obtener:
1. Creación de un Servidor Web 1/5
2. Creación de un Servidor Web 2/5
3. Creación de un Servidor Web 3/5
4. Creación de un Servidor Web 4/5
5. Creación de un Servidor Web 5/5
Traer artículo
Documento: Ejecutado

Figura 1. Muestra en pantalla del formulario de llamada a la aplicación WinCGI de prueba.

miento **EnviarResultadoCGI**, se encargar de abrir el fichero de intercambio (es decir, **CGI_SPWS. OUT**) que habrá rellenado el programa *WinCGI*; se lo envía al cliente; y por último cierra y después borra los archivos de intercambio (**CGI_SPWS.INI** y **CGI_SPWS. OUT**):

```
procedure TForm1.EnviarResultadoCGI (Soc-
      ket: TCustomWinSocket);
begin
  AbrirFichero;
  EnviarFichero (Socket);
  BorrarFichero;
  Socket.Close;
  SysUtils.DeleteFile (NombreFicheroIniCGI);
end;
```

El fichero "ini" es el punto de intercambio de información entre el servidor web y la aplicación WinCGI

ARCHIVOS DEL CD-ROM

El programa WinCGI que desarrollamos en el anterior artículo, que denominamos **CGI_01.EXE**, se adjunta en el CD-ROM de la revista. Deberá situarse en el directorio **CGI-WIN** del directorio raíz, es decir, **c:\home\cgi-win\cgi_01.exe**.

Para llamar al programa **CGI_01.EXE**, se utiliza un formulario web, el cual está contenido en la página HTML **ArticuloServidorWeb.htm** que acompaña también al CD-ROM y cuyo aspecto en pantalla se muestra en la figura 1. Este archivo HTML debe ir en el directorio raíz **c:\home**). Los artículos en HTML, también incluidos, deberán situarse en el directorio **c:\home\articulo\ServidorWeb**. Por último, también se adjunta el código completo del artículo y el programa ejecutable.

CONCLUSIÓN

Aquí terminamos el servidor web que iniciamos hace ya cuatro números. Con él hemos aprendido cómo atender las peticiones de páginas web que realizan los clientes o navegadores, y en concreto, hemos podido comprobar cómo responder a las peticiones de los navegadores "estándares" del mercado: Netscape Navigator e Internet Explorer.

Además hemos añadido a este servidor web la capacidad de diálogo con aplicaciones de servidor (en concreto, de programas *WinCGI*), convirtiéndolo de esta manera en algo activo, capaz de establecer una comunicación cliente-servidor en los dos sentidos posibles de esta comunicación.

El hecho de terminar aquí nuestro servidor web no significa ni mucho menos que tenga una funcionalidad completa, ni siquiera que tenga una función operativa desde un sentido práctico. El motivo es que todavía quedan muchas funciones por definir y muchos matices por perfilar. Por concretar un poco más, se podrían citar los siguientes: soporte completo de las especificaciones *HTTP/0.9* (para lograr una plena "compatibilidad hacia atrás") y *HTTP/1.1*, directorios no cuelguen necesariamente del directorio raíz; definición de una interfaz, similar a *ISAPI* o *NSAPI*, que permita crear aplicaciones de servidor más eficientes (en recursos de memoria y en tiempo de proceso en la CPU); mecanismos de seguridad frente a intrusos, de forma que se dificulte o imposibilite la entrada de *hackers* a la máquina donde se encuentre el servidor web; recuperación ante posibles fallos del propio servidor mediante la creación de archivos de "sucesos" que informen al administrador del sistema no sólo de los fallos y de cuándo se han producido, sino de todos los procesos que se ejecutan y su resultado exitoso o no... y muchos más que a buen seguro se le ocurrirán al lector.

Reconocimiento de voz (II)

Constantino Sánchez Ballesteros (constantino@redestb.es)

En el número anterior de esta serie, comenzamos el estudio de las posibilidades que ofrece el reconocimiento de voz de cara al usuario y dependiendo de la aplicación que se quiera diseñar. Ahora nos adentraremos en los comandos utilizados para la programación en Visual Basic de la API Speech.

El micrófono es el mayor problema que puede encontrar el reconocimiento de voz. De forma inherente, se arrastran con el micrófono otros problemas de muy diversa índole que vamos a detallar:

- No todo el mundo tiene una tarjeta de sonido, aunque actualmente los ordenadores se venden equipados con orientación al mundo multimedia.
- No todo el mundo tiene un micrófono.
- Muchas tarjetas de sonido hacen difícil la instalación de un micrófono cuando no debería ocurrir de este modo.
- La mayoría de los micrófonos que se venden con los ordenadores son baratos. Por ello, la calidad de los mismos es bajísima y permiten la inclusión de ruidos y otros parásitos que imposibilitan al engine el correcto reconocimiento de las palabras.
- Los usuarios no saben utilizar un micrófono. Si éste lleva una solapa en la cabeza, los usuarios se lo ponen incorrectamente, o si el micrófono es de sobremesa, no lo orientan adecuadamente y colocan la boca a una distancia incorrecta (muchos cantantes parece que se comen el micrófono).

La mayoría de las aplicaciones poco pueden hacer por la colocación del micrófono. Por fortuna, cuando el usuario instala un *engine* de reconocimiento, éste le suele ayudar para la correcta colocación de este periférico de entrada.

ERRORES DE LOS ENGINES DE RECONOCIMIENTO

Los *engines* pueden cometer errores, y seguramente, siempre los cometerán (nada ni nadie es perfecto). Lo único que cambia es que cada dos años cometen la mitad de los errores que producían anteriormente.

Para empeorar la situación, los *engines* de dictado que están bien mejorados, y tienen buena gramática, también cometen errores. Desafortunadamente, hay ocasiones en las que los errores de reconocimiento corregidos por el *engine* significan otra cosa muy diferente a la que dijo el usuario.

Una aplicación puede minimizar pequeños errores siguiendo unas pautas determinadas como las que describimos a continuación:

- Hacer más fácil la corrección de errores por parte del usuario.
- Permitir un fácil acceso a la ventana de correcciones para que el usuario pueda corregir los errores que hizo el engine.
- Permitir al usuario el entrenamiento del engine con su voz.

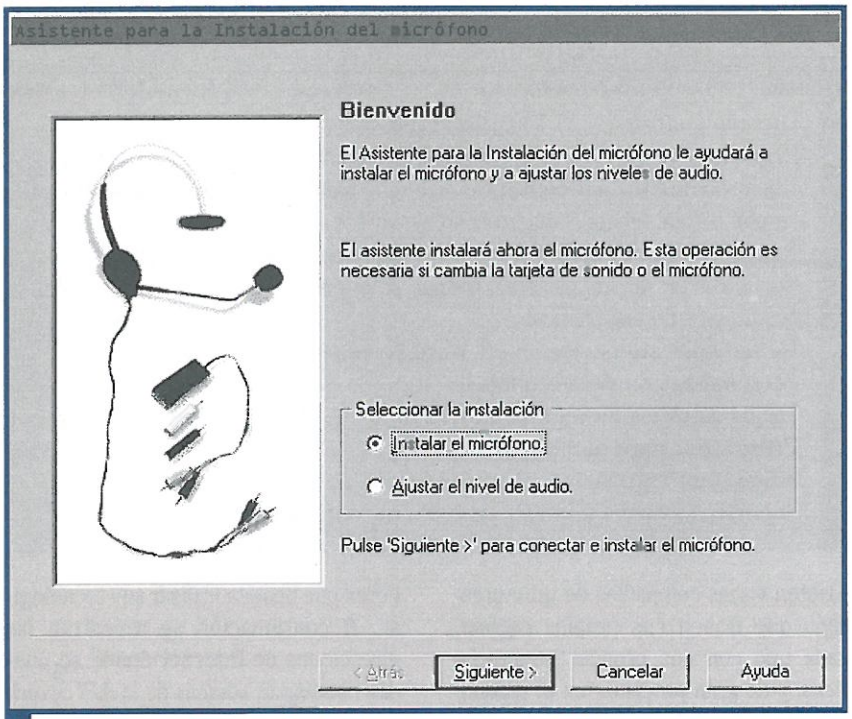

Asistente para la Instalación del micrófono
Bienvenido
El Asistente para la Instalación del micrófono le ayudará a instalar el micrófono y a ajustar los niveles de audio.
El asistente instalará ahora el micrófono. Esta operación es necesaria si cambia la tarjeta de sonido o el micrófono.
Seleccionar la instalación
Instalar el micrófono.
Ajustar el nivel de audio.
Pulse 'Siguiente >' para conectar e instalar el micrófono.
< Atrás
Siguiente >
Cancelar
Ayuda

Figura 1. Detalle de la instalación del micrófono en ViaVoice.

Cuando un *engine* está escuchando para el modo dictado, el usuario puede introducir comandos como "seleccionar palabra" para borrar la palabra previa o capitalizarla. En estos casos, hay que asegurarse de lo siguiente:

- Si un comando sólo tiene una palabra y que ésta no reemplaza a otra palabra que el usuario utilice para dictar.
- Si un comando tiene múltiples palabras que el usuario utilice para dictar.

LIMITACIÓN DEL NÚMERO DE PALABRAS

Los *engines* limitan el número de palabras que están siendo escuchadas desde 20.000 a 100.000 (los nuevos van superando esta cifra). Debido a ésto, una de cada cinco palabras que un usuario dicta no es reconocida porque no está dentro del límite señalado anteriormente.

Los engines limitan el número de palabras que están siendo escuchadas

Las aplicaciones pueden reducir el índice de error de un *engine* si la aplicación le dice a éste qué palabras debería esperar (algo difícil de saber de antemano). Además, hay otro tipo de problemas:

- No tener una lista de palabras corregidas por el usuario es una mala idea porque muchos *engines* reconocerán con poca precisión.
- Que un *engine* no pueda saber quién está hablando, aunque existen algunos que pueden detectar un cambio en el usuario que lo hace. Los algoritmos que existen para el reconocimiento de voz pueden ser utilizados para identificar un usuario, pero actualmente no pueden determinar qué usuario es el que está hablando.
- Un *engine* no puede detectar múltiples usuarios hablando todos a la vez sobre la misma cadena de audio digital.
- Un *engine* no puede oir una nueva palabra y saber qué significa.
- Los usuarios que tengan un acento especial o que hablen dialectos no estándar pueden obtener bastantes errores hasta que entrenen al *engine* para que reconozca sus voces. Esto significa que los *engines* pueden cometer más o menos errores dependiendo de la persona que hable.

CONSIDERACIONES PARA DISEÑAR APLICACIONES CON LA API SPEECH

No es bueno insertar reconocimiento de voz en aplicaciones que se han diseñado para el teclado o el ratón, ya que se obtendría poco beneficio. Después de todo, ¿cuántas aplicaciones de DOS que fueron diseñadas para el teclado son igual de efectivas utilizando el ratón?.

Muchas aplicaciones de dictado permiten dictado discreto, obteniendo del usuario hasta 50 palabras por minuto. Aunque este valor sea aceptable para los

que no saben escribir a máquina, los que sí saben tienen por lo menos 70 palabras por minuto. En efecto, el dictado discreto no será nunca utilizado por buenos mecanógrafos debido a la limitación del *engine*. Por el contrario, el dictado continuo permite hasta 120 palabras por minuto, cifra más que respetable hasta para el mecanógrafo más rápido.

Tabla 1. Características de Viavoice (IBM).

Dictado continuo: 125-140 palabras por minuto.
Precisión: 90-95 % de aciertos en el reconocimiento.
Reconocimiento del contexto: distingue palabras y frases de sonido similar como "qué techo", "que te echo" y "qué te he hecho".
Reproducción en audio: para oír las palabras o frases que se han dictado.
Vocabulario: 42.000 palabras, aunque se pueden añadir otros 20.000 términos propios de uso particular.
Dictado directo en Microsoft Word (versión 7 ó 97)
Personalización del vocabulario
Varios usuarios en un mismo sistema
Preparador para el dictado
Micrófono: especial para el reconocimiento.
Soporte de varios idiomas, incluido el castellano

INFORMACIÓN AL USUARIO

Dado que muchas aplicaciones de hoy en día no incluyen reconocimiento, los usuarios encontrarán esta tecnología como algo interesante y útil a primera vista. En el otro extremo, muchos no asumen que su aplicación tiene reconocimiento de voz o no saben cómo utilizarlo.

Cuando diseñemos aplicaciones de este tipo orientadas a usuarios, es importante comunicarles que la aplicación soporta esta tecnología y cómo pueden beneficiarse de ella. Hay que poner énfasis en lo que el usuario puede o no puede decir por el micrófono, y cómo lo debe hacer.

Es necesario tener un engine instalado en nuestro equipo para poder utilizar la API

¿QUÉ ENGINE INSTALAR?

Para que el reconocimiento de voz pueda funcionar, debe haber un *engine* instalado en el PC. En el mercado existen varias compañías de gran prestigio que tienen sus propios *engines*, cada uno con sus propias características, y de gran precisión en el reconocimiento. Los más famosos pertenecen a IBM (*ViaVoice*) y Dragon Systems (*Dragón Dictate*). Estos son compatibles con la *API Speech*, por lo tanto, cuando programemos una aplicación debe funcionar tanto en un *engine* como en otro, puesto que hay compatibilidad total. En mi caso, he utilizado el *ViaVoice* y ha funcionado a la perfección.

Para poder utilizarlos, hay que comprarlos (lógico), pero otro caso es la distribución del *engine* con nuestras aplicaciones. En ese caso, se debe consultar con la empresa que lo desarrolló para ver los derechos de autor y la licencia de uso. Microsoft ofrece varios *engines* gratuitos que se pueden bajar de Internet junto con la *API Speech*. Lo malo es que sólo utilizan como lengua el inglés (inglés americano) y puede ser un handicap para programadores que quieran dar soporte en español. IBM y Dragon tienen sus *engines* en varios idiomas, entre ellos el español. A continuación, en la tabla 1 y 2, se presentarán algunas características de estos últimos para que juzguéis por vosotros mismos.

Los dos paquetes tienen un software de desarrollo para los programadores que deseen utilizar sus tecnologías. A continuación se muestran las direcciones de Internet donde se pueden conseguir, además de la *API Speech* de Microsoft:

Microsoft:
http://research.microsoft.com/stg/
IBM: http://www.ibm.es
Dragon Systems:
http://www.dragonsystems.com/

Para poder comenzar a trabajar con los listados que vayamos incluyendo en los sucesivos artículos, es necesario bajarse por lo menos la *API Speech* de Microsoft y los *engines* de reconocimiento y texto hablado (todos ellos en la versión 3.0). Están en inglés pero son suficientes para empezar a programar. Si alguno es poseedor de otro *engine* como *ViaVoice* o *Dragón Dictate*, podrá utilizarlo sin problemas.

Aunque los requerimientos para utilizar estos *engines* se sitúan en simples Pentiums (según los fabricantes), es recomendable un Pentium 233 MMX con 64 Megas de RAM (mejor un P-II, por supuesto). Se pueden utilizar compiladores de C++ o Visual Basic 5.0 ya que los listados aparecerán en estos dos lenguajes de programación para dar mayor variedad y alcance a todos los programadores. Hay que tener presente que los componentes

ActiveX utilizados para Visual Basic tienen menos funcionalidades que la programación directa en C++.

OLE AUTOMATION EN VISUAL BASIC

Los objetos *Voice-Command* (Comandos de voz) y *Voice-Text* (texto hablado) son expuestos como objetos *OLE automation*, que permiten ser accedidos y controlados a través de Visual Basic, Visual Basic para aplicaciones (VBA), y otros compiladores que soporten controles ActiveX. A continuación se detallarán los métodos y propiedades que los programadores tenemos disponibles para el soporte de reconocimiento (Comandos de voz) en Visual Basic.

OBJETO VOICE-COMMAND (COMANDOS DE VOZ)

Las siguientes propiedades y métodos descritos se aplican únicamente a este objeto:

PROPIEDAD AWAKE

```
objeto.Awake
```

Activa o desactiva el reconocimiento de voz. Los valores aceptados son *True* para activar y *False* para desactivar.

Ejemplo:

```
Vcmd.Awake = False
```

PROPIEDAD CALLBACK

```
objeto.Callback
```

Especifica el nombre de un módulo de clase que contiene métodos que reciben notificaciones de un objeto *Voice-Command*. El módulo de clase debe contener dos métodos: *CommandRecognize* y *CommandOther*. Estos métodos detectarán los comandos reconocidos o discriminados.

Ejemplo:

```
Vcmd.Callback = "TESTVCMD.VCmdSink"
```

Los métodos *CommandRecognize* y *CommandOther* tienen la siguiente sintaxis:

- **Function CommandRecognize(szCommand as String, dwID as Long)**
- **Function CommandOther (szCommand as String, szApp as String, szState as String)**

En el ejemplo anterior, **"TESTVCMD.VCmdSink"** es una identificación del programa, donde **"TESTVCMD"** es derivado del nombre de proyecto de Visual Basic y **"VCmdSink"** es derivado del nombre de la propiedad del módulo de clase.

PROPIEDAD COMMANDSPOKEN

```
objeto.CommandSpoken
```

Identifica el último comando de voz que se ha reconocido. La aplicación debe llamar al método *MenuCreate* antes de chequear esta propiedad.

Ejemplo:

```
CmdHeard = Vcmd.CommandSpoken
If CmdHeard > 0 Then
   LCommand.ListIndex = CmdHeard - 1
End If
CmdHeard = 0
```

El objeto *Voice-Command* asigna la propiedad *CommandSpoken* al identificador de comandos cuando el usuario habla un comando. Si la aplicación tiene un mecanismo para recibir notificaciones, *CommandSpoken* puede ser ignorado.

Si no hay forma de recibir notificaciones, la aplicación debe chequear continuamente este atributo para averiguar si un comando ha sido hablado. Normalmente, la aplicación debe efectuar el chequeo dos veces por segundo.

Después de que se encuentre un valor válido, se debería establecer a cero el atributo para borrarlo y detectar futuros comandos.

MÉTODO MENUCREATE

```
objeto.MenuCreate (applicationName as String,
      state as String, languageID as Integer,
      dialect as String, flags as Long)
```

Crea un nuevo grupo de comandos de voz para la aplicación. Este método retorna un objeto *Voice-Menu* (menú de voz).

Ejemplo:

```
Dim VMenu As Object
.
Set Vmenu = Vcmd.MenuCreate("My Application", "MyMenu", 1033&, "",
      vcmdmc_CREATE_ALWAYS)
```

Tabla 2. Características de Dragon Dictate.

Dictado continuo
Dictado directo en Word, Wordperfect, Navigator, Explorer, etc.
Vocabulario de 30.000 a 120.000 palabras, según versiones.
Micrófono
Preparadores para el reconocimiento y dictado
Soporte de varios idiomas, incluido el castellano

A continuación se describen los parámetros necesarios para la invocación del métod que son:

APPLICATIONNAME

Nombre de la aplicación — Por ejemplo, "Microsoft Word". Este nombre debe ser único en todas las aplicaciones registradas para que el usuario pueda utilizar los comandos de voz sobre esta aplicación.

STATE

Nombre único del estado de la aplicación para el cual el grupo de comandos de voz es válido. El estado de la aplicación suele corresponder a una ventana activa o cuadro de diálogo.

LANGUAGEID

Identificador del lenguaje. Los primeros 10 bits indican el lenguaje primario, como inglés, francés, español, etc. Los últimos 6 bits son los que indican el "sublenguaje", el cual no es otra cosa que un valor local. Por ejemplo, el portugués tiene dos sublenguajes: brasileño y estándar.

DIALECT

Dialecto específico del lenguaje — Por ejemplo, "Sevilla" o "La Coruña". El parámetro *dialect* puede ser "Standard" (o su equivalente localizado) para el dialecto estándar utilizado por la televisión y radio.

FLAGS

Flag se encarga de indicar cómo crear el menú. Puede utiilizar alguno de los siguientes valores:

Valor
Descripción

Vcmdmc_CREATE_NEW
Crea un menú vacío con el nombre dado. Si existe un menú con ese nombre en la base de datos del *Voice-Command*, el método retornará error. El nuevo menú es guardado en una base de datos cuando el objeto *Voice-Menu* es borrado de la memoria.

Vcmdmc_CREATE_ALWAYS
Crea un menú vacío con el nombre dado. Si ya existe uno con ese nombre en la base de datos del *Voice-Command*, el menú se borrará. El nuevo menú es guardado en una base de datos cuando el objeto se borra.

Vcmdmc_CREATE_TEMP
Crea un menú vacío con el nombre dado. Si existe un menú con el mismo nombre, el método retornará error. Este nuevo menú es temporal, y es ignorado cuando se borra el objeto *Voice-Menu*.

Vcmdmc_OPEN_ALWAYS
Abre un menú existente con el nombre dado. Si el menú no existe, el método creará un nuevo menú vacío. Éste nuevo menú es guardado en la base de datos cuando el objeto *Voice-Menu* es borrado.

Vcmdmc_OPEN_EXISTING
Abre un menú existente. Si el menú no existe, el método retornará error.

MÉTODO REGISTER

```
objeto.Register(siteName as String)
```

Registra una aplicación con el objeto *Voice-Command*. La aplicación llamará a este método antes de llamar a otro o de asignar cualquier propiedad.

Ejemplo:

```
Call Vcmd.Register("")
```

A continuación vamos a dedicarnos a detallar los parámetros que posee el método:

SITENAME

Nombre del lugar a conectar, como "PC Remoto". Para usar el lugar (PC Local), especificaremos una cadena de caracteres vacía.

OBJETO VOICE-MENU

Las propiedades y métodos que se describen a continuación, se aplican al objeto *Voice-Menu*.

PROPIEDAD ACTIVE

```
objeto.Active
```

Activa o desactiva el menú *Voice-Command*. Una aplicación debe llamar al método *MenuCreate* antes de asignar esta propiedad.

Ejemplo:

```
Vmenu.Active = True
```

Cuando la propiedad **Active** es **True**, el menú *Voice-Command* es activado mientras el reconocimiento esté en vigor para el lugar al que hemos conectado (Por ejemplo, nuestro PC). Estableciendo **False** lo que ocurre es que se desactiva el menú *Voice-Command* actual.

MÉTODO ADD

```
objeto.Add(identifier as Long, command as
    String, category as String, description as
    String)
```

Agrega un nuevo comando al menú *Voice-Command* creado mediante el método *MenuCreate*.

Ejemplo:

```
Call Vmenu.Add(CmdNum, "Copy")
```

A continuación se detallan los parámetros del método anterior:

IDENTIFIER

Número único para el comando. La propiedad *CommandSpoken* toma este número cuando el comando es reconocido.

COMMAND

Cadena de caracteres de *Voice-Command*, que por ejemplo: "Abre el fichero".

CATEGORY

Cadena de caracteres que indica la categoría asignada al comando. Los comandos de un menú de voz son organizados en diferentes categorías para ayudar al usuario a navegar a través de la lista de comandos de forma cómoda. Este concepto es similar a los menús de Windows, que organizan comandos bajo nombres de menú como "Fichero", "Editar", "Ver", etc. Para obtener los mejores resultados, deberíamos utilizar 20 o menos categorías de comandos.

DESCRIPTION

Cadena de caracteres que describe la acción realizada por el comando.

PROPIEDAD HWNDMENU

```
objeto.hWndMenu
```

Establece el *handle* de la ventana para el menú de voz. Siempre que la ventana reciba el foco, el menú de voz es automáticamente activado; de lo contrario, se desactiva. Si esta propiedad se establece como **NULL**, el menú será global, es decir, el reconocimiento estará activo hasta que la aplicación lo desactive de forma explícita.

Esta propiedad debería establecerse antes de asignar la propiedad *Active* del objeto *Voice-Menu*. Por defecto, tiene un valor **NULL**.

MÉTODO LISTSET

```
objeto.ListSet(listName as String, numberOfElements as Long, listData as String)
```

Asigna las frases en una lista de comandos de voz. Esta lista es un grupo de palabras o frases que se pueden decir. Enlazando el nombre de una lista en una cadena de comandos, el usuario puede decir cualquier palabra o frase de la lista en un punto determinado por la cadena de comandos. Por ejemplo, una lista llamada "mis amigos" podría contener los nombres Cipri, Iván y Nacho. Si el nombre de la lista fuese enlazado en una cadena de comandos como "Envía mail a <Nombres>", el usuario podría decir "Envía mail a Cipri", "Envía mail a Iván" o "Envía mail a Nacho".

Ejemplo:

```
Dim Names
Dim szNULL as String
szNULL = Chr(0)
.
.
.
Names = "Larry" & szNULL & "Mike" &
        szNULL & "Gib" & szNULL & "Doug"
        & szNULL & "George" & szNull
Call Vmenu.ListSet("name", 5, Names)
```

A continuación se detallan los parámetros del método anterior:

LISTNAME

Nombre de la lista a asignar. Esta lista debe aparecer en la cadena de comandos con al menos un comando en el menú.

NUMBEROFELEMENTS

Número de frases en la lista especificada por *listData*.

LISTDATA

Lista de caracteres. Cada cadena de caracteres es una posible frase para un comando de voz. Utilizando **ListSet** se establece una lista de frases que pueden ser habladas en un comando de voz, como "Envía mail a <nombre>." Típicamente, la lista contiene información que cambia dinámicamente en tiempo de ejecución. Para obtener el mejor rendimiento, la lista no debe superar los 20 elementos, ya que si se supera este valor se reduce el rendimiento en el reconocimiento.

MÉTODO REMOVE

```
objeto.Remove(identifier as Long)
```

Elimina un comando de un menú *Voice-Command* creado por el método *MenuCreate*.

Ejemplo:

```
Call Vmenu.Remove(identifier)
```

A continuación se detallan los parámetros del método:

IDENTIFIER

Identificador del comando a eliminar. Este es el mismo identificador único utilizado por el método **Add** para agregar comandos.

LO PRÓXIMO...

Sé que estaréis deseando comenzar a insertar líneas de código en el compilador, así que un poco de paciencia. La siguiente entrega de esta serie contendrá el primer programa para reconocimiento de voz, concretamente, reconocimiento de comandos hablados por el usuario.

Creación de Componentes (II)

Francisco Rosado Alcántara (fmra@redestb.es)

Continuamos con el desarrollo de componentes en C++ Builder. El mes pasado dejamos en el aire la creación de un componente de forma manual, es decir, sin usar el asistente, por lo que antes de ver a fondo qué son propiedades, métodos y eventos, crearemos un componente de esta forma.

De forma análoga a la creación mediante el asistente, en este caso también debemos seguir una serie de pasos para alcanzar nuestro objetivo:

- 1) Crear la unidad.
- 2) Heredar el componente de otra clase existente.
- 3) Declarar un constructor nuevo para nuestro componente.
- 4) Registrar el componente.

Realmente, desde un punto de vista global, son exactamente los mismos que los empleados en el desarrollo de un componente mediante el asistente del programa.

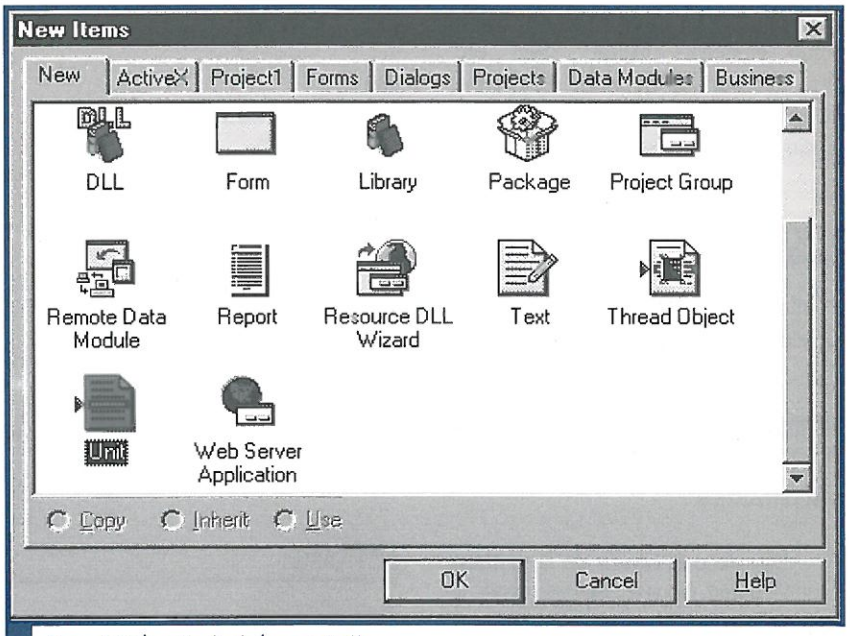


Figura 1. Selección desde la pestaña New.

CREAR LA UNIDAD

Como ya es sabido, una unidad en C++ Builder se compone de un fichero fuente de tipo *CPP* y un fichero de cabecera que al compilarse dan como resultado un fichero *OBJ*. Todos los componentes llevan asociada una unidad. Para crear una nueva unidad, puede elegir alguno de estos dos pasos a los que nos referimos a continuación:

- a) Seleccionar la opción **File** y después **New Unit**.
- b) Elegir la opción **File**, después **New** y por último dentro de la pestaña *New* el componente **Unit**.

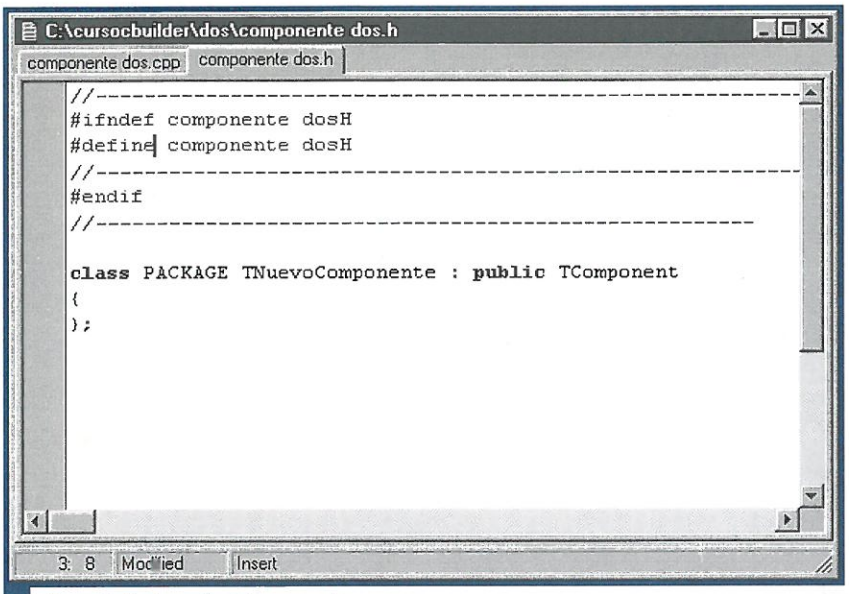


Figura 2. Declaración de la nueva clase.

De este modo lo que conseguimos es crear un fichero de cabecera y el propio fichero *CPP*.

Si lo que queremos es acceder al fichero de cabecera, bastaría con situar el cursor encima del nombre del fichero de cabecera, después pulsar el botón derecho del ratón y a continuación seleccionar **Open File.** El siguiente paso es grabar la unidad con un nombre cualquiera.

DERIVAR EL COMPONENTE

La forma de derivar o heredar una clase de un componente es algo tan simple como añadir la declaración de la clase al fichero de cabecera. Añada las líneas de la figura 2 al fichero de cabecera. Vamos a crear un componente que hereda de TComponent, de igual forma que en el mes pasado con el asistente.

Obviamente es necesario además añadir las clases que deben ser importadas. Por regla general nos bastará con añadir las indicadas en la figura 3.

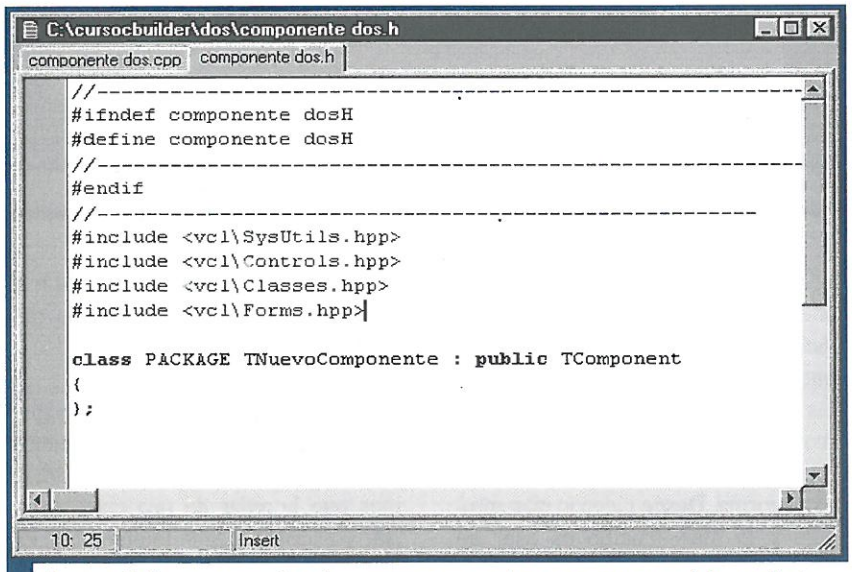


Figura 3. En la imagen se indican las sentencias que incluye necesarias para el desarrollo de casi todos los componentes, así como la declaración de la nueva clase.

DECLARACIÓN DEL NUEVO CONSTRUCTOR

Cada nuevo componente que se desarrolla debe poseer un constructor que redefina el constructor de la clase de la que se ha derivado. La definición del nuevo constructor en la clase sería tal y como aparece en la figura 4.

Y su implementación en el fichero fuente *CPP* se muestra a continuación en la figura 5.

Aprovecharíamos esta implementación para inicializar variables, y efectuar operaciones iniciales de nuestro componente.

REGISTRO DEL COMPONENTE

Al igual que hicimos al crear el componente con el asistente resulta imprescindible registrarlo. Recordemos que su función consiste en indicar al programa qué componentes debemos añadir a la paleta de componentes y en qué paleta debe aparecer, siendo un proceso muy simple. Añadimos una función denominada *Register* al fichero *CPP*, usando un *namespace*, que no es más que el nombre del fichero en el que se encuentra el componente sin la extensión.

Dentro de esta función definiremos un array abierto que contendrá todos los componentes que deseemos registrar de una sola vez. En nuestro caso, lógicamente, contendrá un solo componente. Haremos una llamada a la función *RegisterComponents* por todos y cada uno de los componentes que deseemos registrar.

RegisterComponents es una función con tres parámetros: el nombre de la paleta para el componente, el array de componentes que antes hemos definido y la posición dentro del array de ese componente. Recordemos que los arrays empiezan a numerarse por el 0, con lo que será su posición –1.

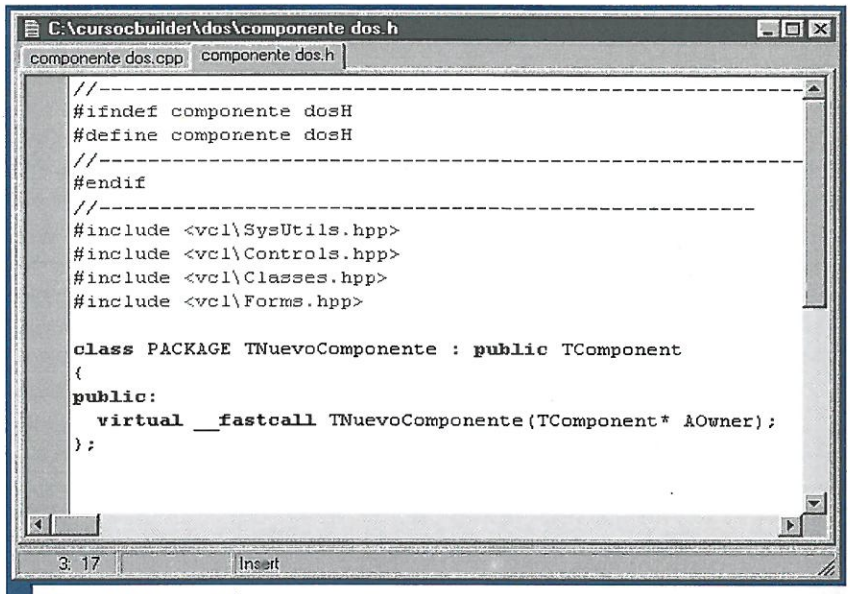


Figura 4. Definición del nuevo constructor

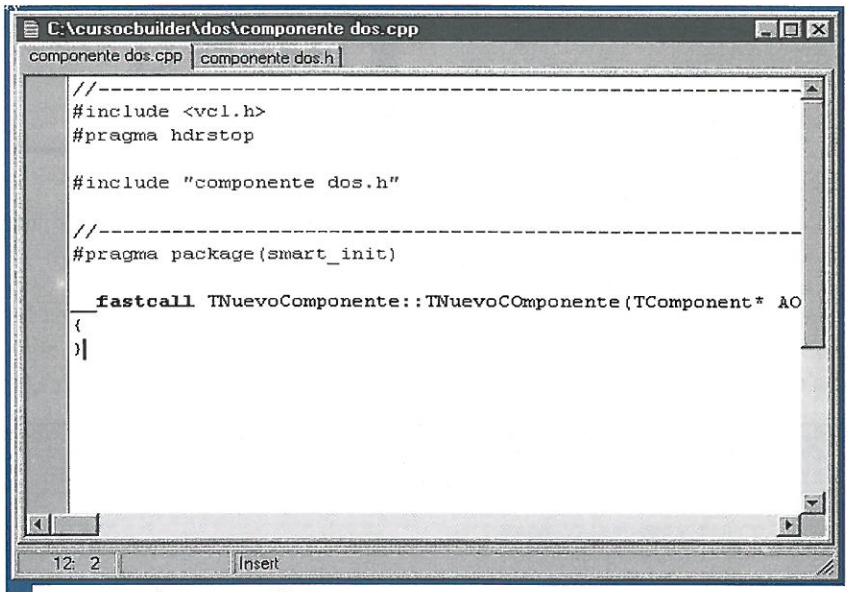


Figura 5. Implementación del nuevo constructor.

NUESTRO PRIMER COMPONENTE

Ya hemos definido todos los elementos necesarios para la confección de un componente, así que vamos a pasar a la práctica. Tal y como se indicaba en el artículo anterior, lo primero que tenemos que pensar es qué va a hacer nuestro componente, cuáles van a ser sus funciones y dónde lo vamos a usar.

Dentro de esta función definiremos un array abierto que contenga los componentes que deseemos registrar

Muchas veces hemos tenido la necesidad de usar un led que se encienda y se apague. Ese va a ser nuestro primer componente. El led tendrá dos posiciones definidas por dos colores diferentes, rojo y verde. Si el led está On, su color será verde y si está Off su color será rojo. El componente tendrá, por tanto, una propiedad On/Off.

El segundo paso que debemos de dar consiste en decidir de dónde vamos a heredar para obtener nuestro componente. Los led suelen tener una forma más o menos redondeada, por lo tanto creo que sería lo más apropiado heredar de un *TGraphicControl* que posea esa misma forma. Desde luego lo más apropiado y lo más sencillo es usar un *TShape* con forma redondeada. Es cierto que un componente *TShape* tiene propiedades y métodos que no necesitamos para nada, pero también es más cierto aún que posee todas las que necesitamos.

Recuerda que no podemos suprimir propiedades de un componente heredado. Si lo deseásemos, tendríamos que heredar de una clase superior y escribir toda la implementación de las propiedades, eventos y métodos, cosa que no suele ser nada recomendable.

Volviendo al componente, parece que lo único que debemos hacer es cambiar la propiedad de *TShape* a la forma que deseemos y cambiar el color del *TBrush* cada vez que cambie nuestra nueva propiedad *On/Off*.

CREANDO UNA APLICACIÓN DE PRUEBA

Una recomendación para todo desarrollador de componentes es la creación de una aplicación de prueba que permita verificar que el código escrito para el componente tiene un funcionamiento correcto. Esto ahorrará tener que compilar la paleta de componentes cada vez que hagamos alguna modificación sobre el componente. La aplicación es tan simple como un formulario con un pequeño botón que permita probar el led. Para ello crearemos una aplicación llamada **LEDProj.bpr** y la unidad **LEDForm.cpp**, en la que introduciremos el nuevo botón desde la paleta de herramientas.

CREACIÓN DEL COMPONENTE

Para la creación del componente, por su rapidez, usaremos el asistente de creación. Recordemos los pasos:

- 1) Seleccione la opción **File**, después **New** y por último **Component**.
- 2) El nombre de la clase será **TLed** y el antecesor de la misma *TShape*, y por supuesto, lo insertaremos en la paleta *Sólo Programadores*.

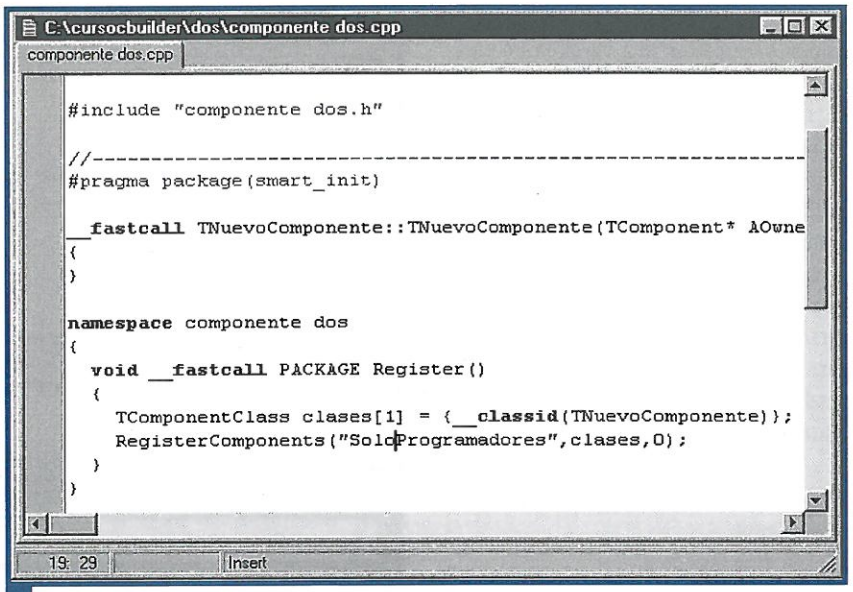


Figura 6. Registro del componente.

- 3) Grabe el componente creado con el nombre **led.cpp**.

Como podemos observar dentro del código que ha generado Builder de forma automática, tenemos tanto la implementación del constructor como el registro del componente, con el *namespace* apropiado, tal y como hemos visto en la primera parte del artículo, pero en este caso ha sido el asistente quien lo ha creado todo. Echemos también un vistazo al fichero de cabecera, en el que se ha añadido la definición de nuestra clase.

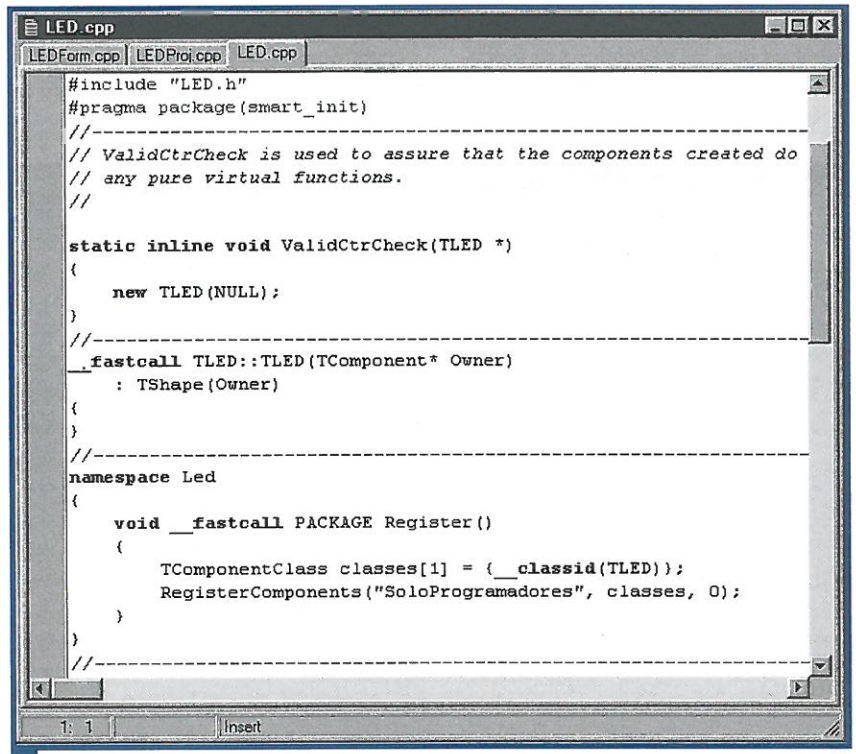


Figura 7. Código generado por C++ Builder.

Como indicamos al principio, vamos a necesitar una variable booleana que indique si el componente está en *On* o en *Off*, así como otras dos variables que almacenen el color que deseemos para los dos estados. C++ Builder ofrece un tipo de variable denominado *TColor* donde podremos almacenarlos. Por lo tanto, definiremos las tres variables y las situaremos en la parte privada de nuestro componente, para que sólo puedan ser utilizadas desde aquí, tal como se muestra a continuación:

```
private:
  bool FOnOff;
  TColor FOnColour;
  TColor FOffColour;
```

Desde luego el nombre de las variables puede ser el que el lector desee, pero convencionalmente se denominan las variables que se van a usar dentro de la clase como propiedades con una letra **F** delante. El siguiente paso consiste en definir los *accessors* y los *mutators* para esas propiedades. Los primeros son funciones que permiten el acceso a los datos de las clases, mientras que los segundos son funciones que permiten modificar los datos de las mismas.

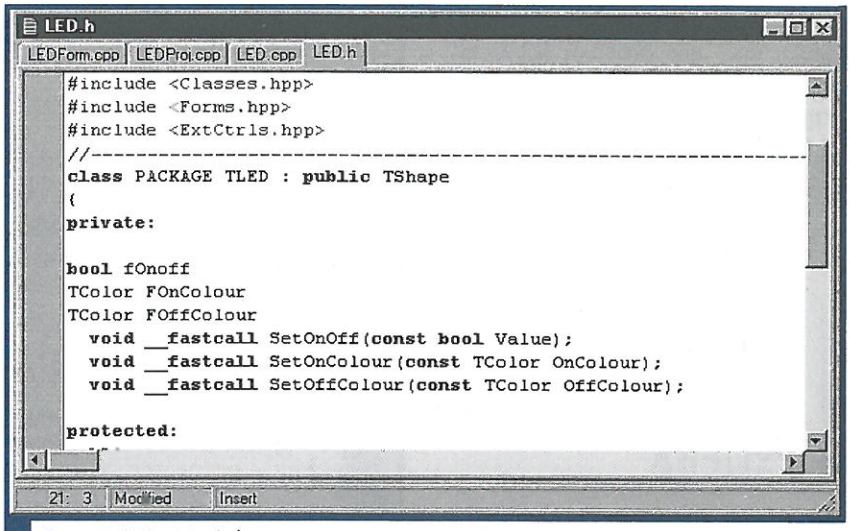


Figura 9. Definición de los accessors y mutators.

Como se observa, hemos definido los mutators en la parte privada de la clase. Parecería una incongruencia que algo que debe ser modificado desde el exterior sea privado. Bien, es cierto, pero como recordaremos del anterior artículo de esta serie, C++ Builder nos ofrece la parte __published, que es donde debemos reflejar las propiedades que se representarán en el Inspector de Objetos del IDE del producto. Para ello añadiremos las siguientes líneas que aparecen en la figura 10 en el fichero de cabecera.

Como observamos todas las propiedades empiezan con la palabra reservada __property, seguida por el tipo de variable que usa y las variables o funciones de lectura y escritura. Sí, como podemos observar *read* está asociada a una variable y *write* a una función. Lógicamente ambas pueden apuntar a donde se desee, variable o función, pero es altamente recomendable que *write* apunte a una función donde nosotros hagamos las comprobaciones oportunas sobre la validez del dato introducido.

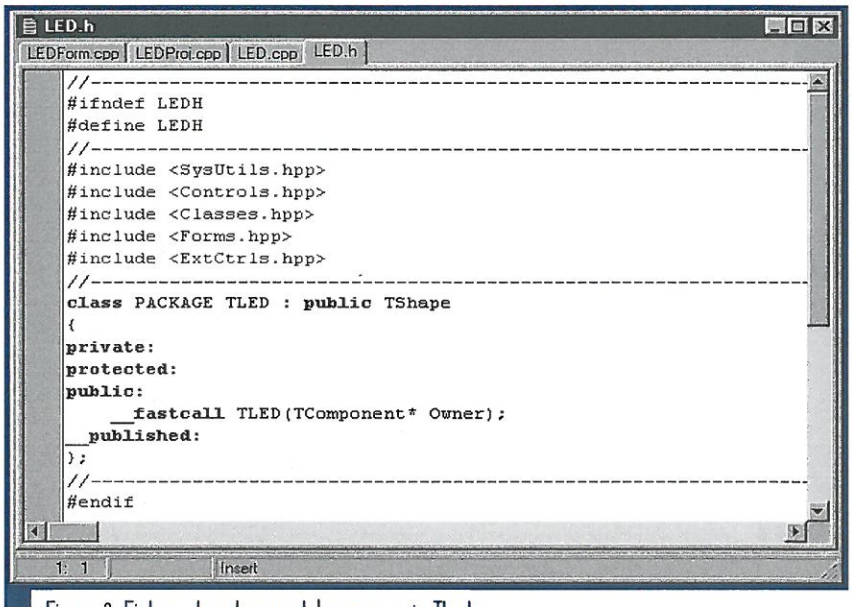


Figura 8. Fichero de cabecera del componente TLed

Los mutators se encargan de modificar los datos, convirtiéndose en la parte privada de la clase

De las posibles variaciones de los valores de las propiedades hablaremos en otra entrega del curso. Sería recargar demasiado nuestro primer componente que ante todo debe ser muy simple. De todos modos, con las definiciones efectuadas en el fichero de cabecera, prácticamente hemos agotado la parte dura de nuestro componente. Sí, es cierto, lo más fácil es

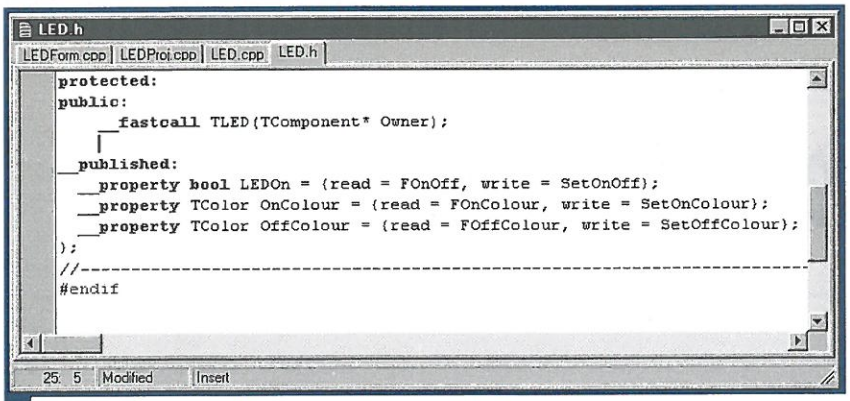

LED.h
LEDForm.cpp | LEDProj.cpp | LED.cpp | LED.h
protected:
public:
__fastcall TLED(TComponent* Owner);
__published:
__property bool LEDOn = {read = FOnOff, write = SetOnOff};
__property TColor OnColour = {read = FOnColour, write = SetOnColour};
__property TColor OffColour = {read = FOffColour, write = SetOffColour};
};
#endif
25: 5 Modified Insert

Figura 10. Declaración de las propiedades en _published.

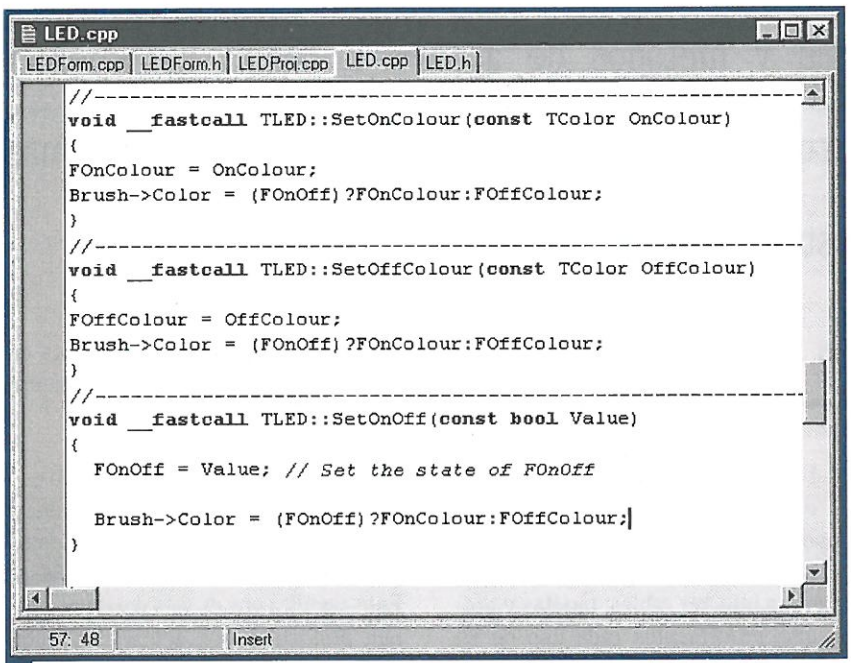


Figura 11. Definición de los mutators.

la implementación de las funciones de escritura.

Volvamos al fichero *CPP* y escribamos los *mutators*, tal y como se refiere en le figura 11.

El siguiente paso será escribir el código relativo al constructor. En el constructor reflejaremos todas las propiedades que deseemos aparezcan por defecto en el *Inspector de Objetos*.

Pues ya está todo. Sólo nos queda probar que el componente que hemos desarrollado funciona bien, y que cumple con su función. Para ello vamos a usar la aplicación de prueba que hicimos al principio, donde añadiremos el componente desarrollado al proyecto. Para ello vayamos al fichero **LEDForm.Cpp** y en su fichero de cabecera añadamos nuestro componente.

Añade en la parte privada del fichero de cabecera LEDForm.h el siguiente código:

```
TLED* LED1;
```

Volvamos al formulario y vamos a escribir código para el evento *OnCreate*.

```
void _fastcall TForm1::FormCreate(Tobject
      *Sender)
{
LED1 = new TLED(this);
LED1-> Parent = this;
LED1->Left  =  (Width/2)-(LED1->Width/2);
LED1->Top = (Height/2)-(LED1->Height/2);
}
```

Este código crea dinámicamente un componente *TLed* y lo sitúa en el centro de la pantalla.

En el constructor reflejaremos todas las propiedades que deseemos aparezcan por defecto en el Inspector de Objetos

Por último, asociaremos el código al evento OnClick del botón, para cambiar las propiedades de nuestro led, tal y como se muestra a continuación:

```
void _fastcall TForm1::Button1Click(Tobject
      *Sender)
{
LED1->LEDOn=!LED1->LEDOn;
}
```

Ha llegado el momento de probar nuestro componente. Pinche en el botón **Run** y si todo es correcto, aparecerá un pequeño led de color rojo en el centro de la pantalla. Pulsando sobre el botón podremos cambiar el color de nuestro led.

Sólo falta añadir el componente al paquete de componentes que creamos en el primer artículo del curso, y el proceso habrá terminado.

A lo largo del próximo número nos dedicaremos a hacer un componente ya un poco más serio, y aprenderemos a cambiar el icono del componente, el que aparece en la paleta de componentes, a través de un pequeño fichero de recursos.

Cableado de red

Enrique Díaz Trobo (enriqued@jet.es)

Este mes abordamos el tema del cableado en las redes locales. Veremos también algo sobre topología de red y métodos de acceso al cable. Finalmente, abordaremos el cableado estructurado, cuyos métodos nos pueden ahorrar más de un dolor de cabeza si nuestra red crece rápidamente.

Las redes locales no son otra cosa que sistemas de comunicación. Aunque sean pequeñas y resuelvan situaciones muy particulares, ello no quiere decir que lo tengamos que hacer peor que los grandes. Como tales sistemas de comunicación, las redes han de ser modulares y adaptables para poder adecuarse a muchas necesidades de ubicación diferentes. La modularidad nos facilitará la incorporación de nuevos elementos o bien la reubicación de los ya existentes.

OSI es un estándar de protocolos de comunicaciones

También deberemos tener en cuenta otros mil pequeños detalles como es cual es el tipo de cableado más adecuado a cada situación, su coste, etc. Para poder decidir estas cosas veremos diferentes tipos de cables, de topologías y también qué es eso del cableado estructurado.

OSI

El párrafo anterior nos sugiere que deberíamos utilizar varios cables y topologías para adecuarnos a cada situación, lo cual quiere decir que vamos a integrar cables, tarjetas y protocolos provenientes cada uno de su padre y de su madre. Un lío. Para desenmadejar tal lío tenemos (gracias a Dios) algo que se llama OSI (interconexión de sistemas abiertos).

OSI es un estándar de protocolos de comunicaciones que nos ayudará a crear sistemas de red interoperativos. Para conseguirlo, OSI define una estructura multinivel que diferencia cada uno de los procesos de comunicación requeridos para transmitir datos entre diferentes redes, sistemas y aplicaciones.

Esta estructura (la famosa pila OSI) parte de los niveles más bajos, esto es, los físicos, así se definen estándares para conectores y cables, en cuanto a las señales eléctricas que utilizarán para la transmisión de bits de datos. Como podéis ver, OSI comienza desde el principio.

Sobre este nivel tenemos otro que se encarga del enlace de datos, es decir, sobre cómo se 'empaquetan' los datos para transportarlos sobre la red física. Esto significa que es aquí donde se definen protocolos que actuarán directamente sobre los componentes físicos de la red, más concretamente, sobre las tarjetas de red. Se controla el flujo de datos que atraviesa en enlace y establece métodos de verificación de errores en los envíos de estos paquetes.

Para llevar a cabo estas tareas, se definen dos subniveles en nivel de enlace:

- Subnivel de control de acceso. En este nivel es donde se define cada una de las distintas topologías de red, como por ejemplo Ethernet. Todas las especificaciones sobre la forma en que una tarjeta de red se direcciona en el cable se manejan aquí.

- Subnivel de control lógico de enlace (LLC). Establece cómo se ha de desviar un paquete si un nodo tiene más de una tarjeta y éstas están conectadas a topologías diferentes.

Dos protocolos que actúan a nivel de enlace de datos y que se han convertido en un estándar de hecho son:

- ODI (*Open Datalink Interface*). Es el estándar de Novell y ofrece una excelente manera de ejecutar dos o más protocolos de comunicación en un ordenador que tenga múltiples tarjetas de red. Es esta característica la que permite tener una tarjeta física y múltiples tarjetas lógicas. De modo que podemos tener, por ejemplo, los protocolos TCP/IP e IPX montados sobre una única tarjeta de red.

- NDIS es el estándar de Microsoft equivalente al ODI de Novell.

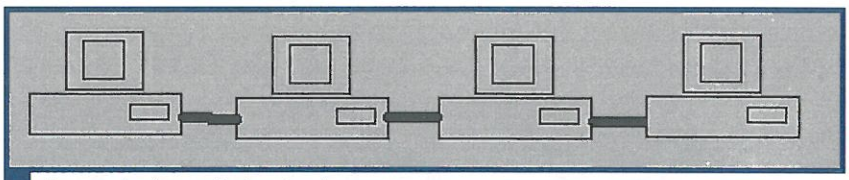


Figura 1. Configuración de red en bus.

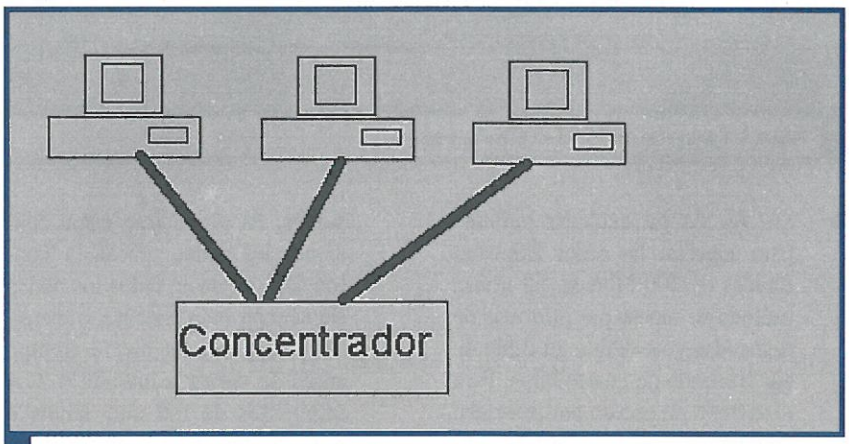


Figura 2. Configuración de red en estrella.

IEEE

El IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) se ocupa de definir unas normas estándares para los diferentes tipos de redes. Para tratar de redes locales en particular se creó la comisión 802, de ahí que los nombres de los protocolos de red tengan el formato de nombre 802.*x*. El estándar de IEEE que empleemos en un segmento de red nos está definiendo, de hecho el tipo de red que estamos empleando en ese segmento. Veamos a continuación alguno de los estándares más comunes:

- 802.1. Define la relación del conjunto de estándares 802 con el modelo de referencia de interconexión de sistemas abiertos, (el OSI del que hablábamos al principio). No es un protocolo en sí mismo, pero sí sienta algunas bases para el resto de los protocolos. Una de estas bases es, por ejemplo, que la dirección de una tarjeta de red estará formada por 48 bits. De esta forma podremos tener una dirección única para cada tarjeta de red. En realidad, la dirección o identificador único de la tarjeta se graba en sus chips y es un número de 48 bits que se genera aleatoriamente. Por tanto, es teóricamente posible que se nos repitiera alguna dirección, lo cual sería realmente mala pata. En cualquier caso es algo prácticamente imposible, ya que una parte de la dirección identifica al fabricante de la tarjeta. Aún no se ha producido el caso de una dirección que se repita, al menos que se sepa. En el caso de las redes ARCNET las direcciones son definibles por el usuario, con lo que se elimina este peligro potencial.

- 802.2 Es el protocolo LLC que describíamos antes. Si algún nodo de nuestra red tiene más de una tarjeta de red, y estas tarjetas se conectan a topologías distintas, cuando se envíe un paquete y éste descienda por la pila de protocolos será desviado a la tarjeta de red correcta cuando alcance el subnivel de control lógico de enlace (LLC).

- 802.3 Este estándar define el protocolo para acceso al cable en modo múltiple mediante detección de portadora y detección de colisiones empleado en Ethernet.

- 802.5. Este protocolo fue desarrollado por IBM para su estándar Token Ring. Utiliza un método de acceso mediante paso de testigo y se trata básicamente de una topología en estrella que forma un anillo lógico.

- 802.9 Se trata de un estándar para la integración de datos y voz para las redes LAN de tipo 802 y redes digitales de servicios integrados (RDSI). La definición de los nodos en esta especificación incluye codificadores/decodificadores de voz y vídeo.

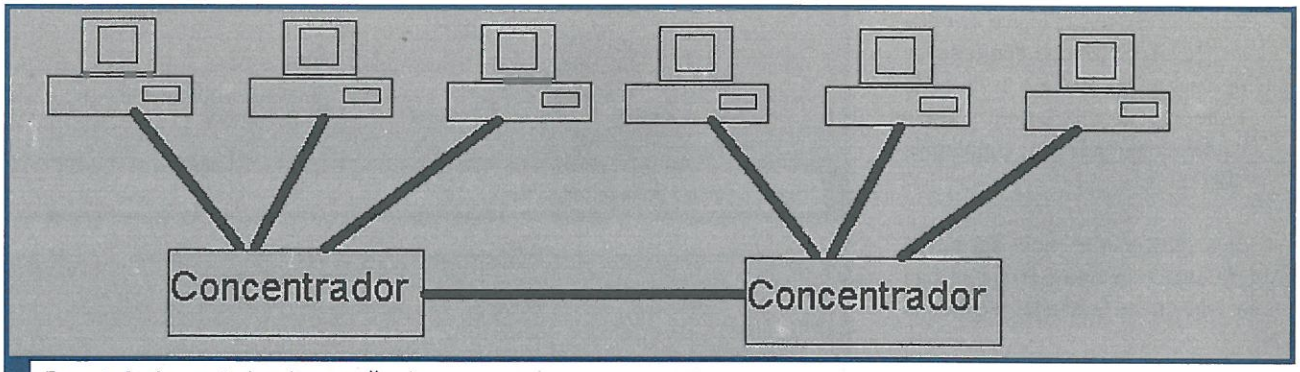


Figura 3. Configuración de red en estrella y bus.

- 802.12. Es un estándar definido para soportar las redes Ethernet rápidas de 100 Mb/seg. Se utiliza método de acceso por prioridad de demandas y se define un cable de par trenzado de cuatro hilos. Para el método de acceso por prioridad de demandas es necesaria la utilización de un concentrador que controle el acceso al cable.

MÉTODOS DE ACCESO

Hemos hablado antes de los métodos de acceso al cable de cada protocolo. ¿Por qué es tan importante esto?, porque sólo tenemos un cable con acceso múltiple, es decir, un solo cable al que acceden muchos nodos. El caso es que sólo uno de los nodos puede transmitir en un momento determinado. Para solucionar este problema de tráfico tenemos los métodos de acceso:

- Acceso por detección de portadora/detección de colisiones (CSMA/CD). Con este método los nodos 'escuchan' la señal de portadora del cable y envían cuando no hay otros nodos transmitiendo. A veces varios nodos detectan que se pueden enviar datos y efectúan el envío al mismo tiempo, con lo cual se producen colisiones. Al producirse estas colisiones los nodos aplican la 'contención', es decir, todos los nodos abandonan su intento por transmitir y esperan un cierto tiempo antes de volver a intentarlo. Los segmentos de red muy largos o muy activos tienen muchas de estas colisiones, con lo cual es conveniente segmentar las redes para reducir la contención.

  Este tipo de acceso es el más rápido. Sin embargo, durante cargas intensas de tráfico las colisiones pueden llegar a congestionar la red y hacerla prácticamente inoperativa.

- Paso de testigo. El testigo es un paquete especial en las redes Token Ring, Token bus y FDDI (fibra óptica). Los nodos toman el paquete testigo, lo que les da 'derecho' a transmitir. Como sólo una estación puede tener el testigo cada vez, el resto de los nodos no intentará transmitir, pues les falta el testigo para hacerlo. De este modo se evita el efecto contención de las redes Ethernet.

- Acceso por prioridad de demandas. Este método nació con las nuevas redes Fast Ethernet. Se trata de quitar a los nodos la gestión del acceso para pasársela al concentrador. Los nodos piden permiso para transmitir con una prioridad y el concentrador transmitirá primero aquellas con mayor prioridad.

TOPOLOGÍAS DE RED

La forma en que conectamos los diferentes ordenadores de la red conforma la topología de ésta. Las topologías básicas que podemos adoptar en una red son:

- Bus (figura 1). Consiste en un sólo tramo de cable que conecta todas las estaciones, formando una especie de cadena. Los paquetes llegan a todas las estaciones, pero sólo los *recoge* aquella estación a la que va dirigido. Para este tipo de red se utiliza Ethernet 802.2 y 802.3 con cable coaxial fino RG58. Tiene el inconveniente de que si la red se rompe en un punto, por la desconexión accidental de un cable, por ejemplo, todos los nodos de la red quedan desconectad.

- Estrella (figura 2). Cada estación se conecta individualmente a un concentrador, que es el que reparte los paquetes al resto de los nodos, individualmente o en su conjunto, según sea necesario. Al contrario que la red en bus, el único punto débil de la red es el propio concentrador, ya que si es un nodo el que falla, el resto no se verá afectado. En esta topología, la opción más barata que podemos encontrar consiste en utilizar cable de par trenza-

do 10Base-T, y si necesitamos más prestaciones, 100Base-T.

- Anillo. En una red en anillo las tramas pasan de un nodo a otro en círculo.

Las topologías en bus y en estrella pueden combinarse. Podemos, por ejemplo, tener dos o más redes en estrella y conectar los concentradores en bus (figura 3). O bien conectar estos concentradores en estrella a otro concentrador central que sería el que gestionaría el tráfico de interconexión entre redes. Esto último sería lo que se llama una estrella jerarquizada (figura 4).

Para grandes redes de interconexión tenemos, además, las siguientes topologías:

- Backbone. Se utiliza habitualmente en edificios. Existe un enlace central al que se conectan todos los departamentos y el tráfico de interconexión de redes se gestiona a través de *routers* y *brigdes*. Por ejemplo, el cableado vertical de un edificio puede ser de fibra óptica y luego conectaremos a éste enlace los concentradores de los distintos departamentos.

- Estrella Jerárquica. Lo comentamos anteriormente. Se trata de conectar los concentradores departamentales a un concentrador central que gestiona el tráfico de interconexión de red.

- Malla. Básicamente se trata de una topología de bus con conexiones redundantes. De este modo si un nodo falla o una de las rutas posibles está congestionada, se puede emplear otro camino. Tiene el inconveniente de ser caro y es difícil planificar el número exacto y conveniente de conexiones.

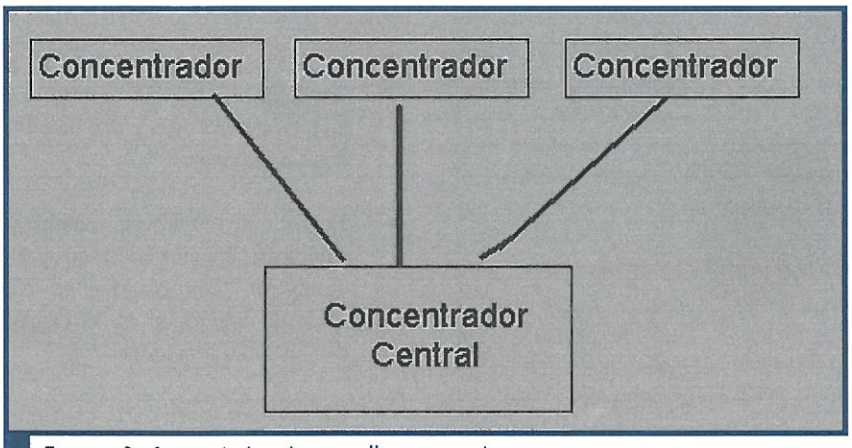


Figura 4. Configuración de red en estrella jerarquizada.

CABLEADO ESTRUCTURADO

El cableado estructurado es un invento relativamente reciente, pensado para dar cabida a futuros servicios, de modo que se faciliten futuras reubicaciones y configuraciones. También deberá dar soporte a productos y entornos de múltiples fabricantes.

Tal como están evolucionando las cosas, tener un sistema fiable de cableado para las comunicaciones es tan importante como tener un suministro de energía eléctrica en el que se pueda confiar.

Hasta no hace mucho, los sistemas 'de toda la vida' eran aceptables, pero, en el mercado actual, ávido de información, nos devora cualquier ancho de banda. El poder proporcionar comunicaciones de voz y de datos por medio de un sistema de cableado estructurado universal es un requisito básico de los negocios que utilicen tecnologías punteras. Estos sistemas de cableado estructurado proporcionan la base sobre la que se puede construir una estrategia general de los sistemas de información.

Un Sistema de Cableado Estructurado nos debe proporcionar:

- Conectores para conexiones provisionales de cables
- Salidas de información
- Cableado Horizontal
- Productos para interconexión de redes
- Cableado principal

En un sistema de este tipo, cada estación de trabajo se conecta a un punto central utilizando una topología de tipo estrella jerarquizada, facilitando así la interconexión y la administración del sistema. Esta disposición permite la comunicación con prácticamente cualquier dispositivo, en cualquier lugar y en cualquier momento. Un plan de cableado bien diseñado puede incluir distintas soluciones de cableado independiente, utilizando diferentes tipos de medios, e instalarlos en cada estación de trabajo para acomodar los requerimientos de funcionamiento del sistema a cada caso.

En cuanto a la topología nuestra estrella jerarquizada ha de cumplir los siguientes requisitos:

- Cada salida de del área de trabajo de telecomunicaciones debe estar conectada directamente al cuarto de telecomunicaciones excepto cuando se requiera para hacer la transición a cable UTC.

- No se permiten correspondencias (múltiples apariciones del mismo par de cables en diversos puntos de distribución) en cableados de distribución horizontal.

- Algunos equipos requieren componentes (por ejemplo puertos RS-232) en la salida del área de telecomunicaciones. Estos componentes deben instalarse externamente a la salida del área de telecomunicaciones. Esto garantiza la utilización correcta del sistema de cableado estructurado.

COMPONENTES DEL CABLEADO ESTRUCTURADO

Los conectores para las conexiones provisionales de cables interconectan los puertos del panel conmutador y conectan el equipo de las estaciones de trabajo a las salidas de datos. Los conectores para las conexiones provisionales de cables facilitan, agregar o cambiar conexiones.

SALIDAS DE INFORMACIÓN

Las salidas de información son los puntos de terminación para los cables que están cerca de la estación de trabajo. Se clasifican de acuerdo al lugar físico de instalación (montaje empotrado, montaje en superficie, modular, piso elevado), la cantidad de puertos por salida, y los tipos de conectores requeridos.

CABLEADO HORIZONTAL

El cableado horizontal comprende el sistema de cableado que se extiende desde el área de trabajo de telecomunicaciones hasta el cuarto de telecomunicaciones y es el medio por el que se transmiten los servicios de comunicaciones.

El cableado horizontal puede ser un cable no blindado con par trenzado (UTP), un cable blindado con par trenzado (STP), o un cable de fibra óptica. Debemos elegir el tipo de cable en función de nuestros requerimientos de trabajo y de ancho de banda.

El cableado horizontal consta de dos elementos básicos:

- Cableado Horizontal y Hardware de Conexión. Proporcionan los medios para transportar señales entre el área de trabajo y el cuarto de telecomunicaciones.
- Rutas y Espacios Horizontales (o sistema de distribución horizontal). Las rutas y espacios horizontales son utilizados para distribuir y soportar cable horizontal y conectar hardware entre la salida del área de trabajo y el cuarto de telecomunicaciones. Estas rutas y espacios son los "contenedores" del cableado horizontal.

El cableado horizontal consta de otro grupo de elementos que pasamos a describir:

- Las salidas (cajas/placas/conectores) de telecomunicaciones en el área de trabajo.
- Cables y conectores de transición instalados entre las salidas del área de trabajo y el cuarto de telecomunicaciones.
- Regleta de conexión (patch) y cables de conexión utilizados para configurar las conexiones de cableado horizontal en el cuarto de telecomunicaciones.

El cableado horizontal contiene más cable que el cableado del backbone y es menos accesible que éste.

Listado 1. Tipos de cables más utilizados.

Cable	Comentario
100Base-T	Fast Ethernet. Soporta hasta 100 Mb/seg.
100VG-AnyLan	Igual que la Fast Ethernet, utilizando un nuevo método de acceso por prioridad de demandas
10Base-2	Cable coaxial fino (RG58) con una longitud máxima de segmento de 185 metros
10Base-5	Cable coaxial con una longitud máxima por segmento de 500 metros.
10Base-F	Fibra óptica con una longitud máxima de segmento de 4.000 metros a 10 Mb/seg.
10Base-T	Cable de par trenzado, con una longitud máxima de segmento de 100 metros
10Broad-36	Cable coaxial (RG59) con una longitud máxima de segmento de 3.600 metros
Fibra óptica	Con la fibra óptica siempre se están batiendo récords. Mediante fibra óptica se puede llegar a alcanzar hasta un millón de Mbps y distanciar los repetidores 100 Km entre sí.

Listado 2. Longitudes máximas de segmento.

Topología empleada	Longitud de segmento
Arcnet coaxial en bus	305 metros
Arcnet coaxial en estrella	609 metros
Arcnet par trenzado en bus	122 metros
Arcnet par trenzado en estrella	122 metros
Ethernet coaxial fino	185 metros
Ethernet de par trenzado	100 metros
Token Ring	100 metros

CONSIDERACIONES DE DISEÑO

Los costos en materiales, mano de obra e interrupción del trabajo por estas labores al hacer cambios en el cableado horizontal pueden ser bastante altos. Es por ello que para evitar estos costos tan elevados, el cableado horizontal debe ser capaz de manejar una amplia gama de aplicaciones de usuario. La distribución horizontal debe diseñarse para facilitar el mantenimiento y el realojamiento de áreas de trabajo.

Esto implica:

- Dar soporte a comunicaciones de voz (teléfono).
- Comunicaciones de datos.
- Redes de área local.

También se debe considerar incorporar otros sistemas de información del edificio (como por ejemplo otros sistemas como la televisión por cable, control ambiental, seguridad, audio, alarmas y sonido) a la hora de seleccionar y diseñar el cableado horizontal.

DISTANCIAS

La distancia horizontal máxima es de 90 metros independientemente del cable utilizado. Esta es la distancia desde el área de trabajo hasta el cuarto de telecomunicaciones. Al establecer la distancia máxima se hace la previsión de 10 metros adicionales para el cable necesario para conectar equipo en el área de trabajo y el cuarto de telecomunicaciones.

TIPOS DE CABLE

Los tres tipos de cable reconocidos por ANSI/TIA/EIA-568-A para distribución horizontal son:

- 1. Par trenzado, cuatro pares, sin blindaje (UTP) de 100 ohmios, 22/24 AWG
- 2. Par trenzado, dos pares, con blindaje (STP) de 150 ohmios, 22 AWG
- 3. Fibra óptica, dos fibras, multimodo 62.5/125 mm

El cable a utilizar preferentemente es el par trenzado sin blindaje UTP de cuatro pares. El cable coaxial de 50 ohmios se acepta pero no se recomienda en instalaciones nuevas.

CABLEADO PRINCIPAL

Un sistema de cableado estructurado consiste en cables horizontales de distribución independiente, conectados por medio de productos para interconexión al cableado ascendente o cableado principal.

El cable principal parte del punto principal de distribución y se interconecta con todas las salidas de telecomunicaciones. Los cables principales están hechos habitualmente de fibra óptica o de cobre con pares múltiples.

LOS CABLES

Ya metidos en harina con tanto cable vamos a hablar un poquito de ellos. Determinar el tipo de cable a emplear es una decisión muy importante, pues es una compra cara y no podemos fallar en eso.

El cable que más proyección tiene hoy en día es el de fibra óptica pero hoy por hoy es caro y difícil de instalar. Sin embargo, es recomendable su utilización para enlazar redes distantes y para crear enlaces muy rápidos entre servidores o interconexión de redes.

El cable de cobre es barato y nos vendrá bien en la mayoría de los casos. En el listado 1 podéis encontrar los tipos más comunes de cables, y en el listado 2 podéis ver las longitudes máximas de segmento.

EL LIBRO OFICIAL DE DESARROLLO CON MICROSOFT INTERNET EXPLORER 4 SITE BUILDER

Microsoft Internet Explorer 4 significa disponer de sitios web espectaculares, aquí se indica cómo crearlos. Si ya conoce todo lo necesario para crear páginas web, si ya utiliza HTML y ahora lo que desea es crear impresionantes sitios web aproveche al máximo las características de Internet Explorer 4. Esta importante guía, escrita por un autor líder en ventas de libros de Internet, Alan Simpson, comienza con una revisión del HTML básico y una introducción a la programación. A continuación construye sobre esta base para mostrarle cómo manipular objetos en las páginas web.

Además de todo esto, este manual también le ofrece consejos de experto y le muestra multitud de ejemplos que le permitirán la creación de páginas con información completa e interactiva.

En el libro encontrará temas tan candentes cómo son el HTML dinámico, hojas de estilos en cascada, VBScript, ActiveX, enlace de datos.

Además va acompañado de un CD-ROM que incluye Internet Explorer 4.01 y el kit de Desarrollo Software (SDK) de Microsoft Internet Client, así como numerosos ejemplos para crear páginas web.

Editorial: McGraw Hill	Autor: Alan Simpson
Nº páginas: 625	Idioma: Español
Nivel: Intermedio-avanzado	Precio: 7.500 Ptas.(I.V.A. inc.)

3D STUDIO MAX 2.0

Con este libro podrá ampliar sus conocimientos sobre 3D Studio MAX, a lo largo del manual se describen sus características centrándose en algunas de las más avanzadas técnicas que ofrece.

Se comienza con el modelado, la base de toda escena o animación, estudiando tanto las ayudas al dibujo que MAX ofrece, como la creación de primitivas estándar. El trabajo con herramientas de deformación, o con las herramientas de creación más sofisticadas, como Nurbs o parches, nos ayudará a redondear tanto el conocimiento del programa a este respecto como las obras que se realicen con él.

El siguiente paso será tratar las texturas, intentando dar una nueva perspectiva a estas, un sentido dentro de la escena. Se conocerán los niveles más sofisticados que el programa permite. Se tratarán temas tan interesantes como el de la iluminación y al final se aprenderá a darle vida mediante las animaciones.

Editorial: Anaya Multimedia	Autor: Javier López Escribá
Nº páginas: 384	Idioma: Español
Nivel: Intermedio-avanzado	Precio: 3.295 Ptas.(I.V.A. inc.)

A FONDO MICROSOFT VISUAL INTERDEV

Si quiere construir sitios web de alto rendimiento o crear intranets, el sistema de desarrollo web de Microsoft Visual InterDev es la respuesta.

Con Visual InterDev, obtendrá sofisticadas opciones de conectividad de bases de datos, capacidad de gestión de sitios, y una estrecha integración con otras herramientas de Microsoft como el sistema de programación Visual Basic, el sistema de desarrollo Visual C++ y el software de desarrollo Visual J++. En pocas palabras, es el software que le permitirá crear y gestionar aplicaciones web avanzadas, manejadas por bases de datos.

Este libro le muestra cómo poner en funcionamiento estas potentes características de Visual InterDev: Controles y Scripts ActiveX en el cliente y en el servidor, Active Server Pages, integración con bases de datos y herramientas de gestión, amplio soporte de lenguajes, etc.

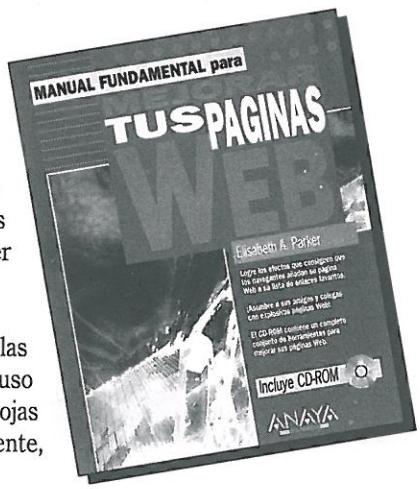
El libro incluye un CD-ROM que contiene una versión de evaluación de Visual InterDev, Internet Explorer 4.01, Design-Time ActiveX Control SDK y numerosos ficheros de ejemplo.

Editorial: McGraw Hill
Nº páginas: 516
Nivel: Avanzado
Precio: 7.500 Ptas.(I.V.A. inc.)

Autor: K. Miller, K. Spencer, E. Vincent y N. D. Evans
Idioma: Español

MANUAL FUNDAMENTAL PARA MEJORAR TUS PÁGINAS WEB

Este libro y el CD-ROM que le acompaña ponen en sus manos todas las herramientas y técnicas necesarias para que pueda desarrollar sofisticados diseños, efectos especiales personalizados, marcos, applets Java, animaciones GIF y efectos multimedia.



Mediante instrucciones claras y ejemplos prácticos se nos enseña cómo sacar partido de todo el freeware y shareware disponible, desde tipos de letras hasta programas de desarrollo web, para empezar a remodelar nuestras páginas web desde el primer momento.

Contiene las herramientas correctas para mejorar sus páginas web, optimice las paletas de colores de sus páginas, encuentre imágenes y archivos multimedia de libre uso en la web, cree gráficos únicos y efectos especiales, practique con tablas, marcos y hojas en cascada, incluya formularios en sus páginas web y cree applets Java instantáneamente, implemente GIFs animados y sonidos en sus páginas.

El CD-ROM que acompaña al libro contiene más de 60 populares herramientas de desarrollo web en modalidad freeware, shareware o versión de evaluación.

Editorial: Anaya Multimedia
Nº páginas: 496
Nivel: Intermedio

Autor: Elisabeth A. Parker
Idioma: Español
Precio: 3.750 Ptas.(I.V.A. inc.)

Correo del lector

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Pregunta

Les escribo este correo para solicitarles una respuesta más concreta en torno a una duda del número 48 de la revista en la que preguntaban si Linux está preparado para el efecto 2000. La respuesta es que sí y ésto no plantea problemas.

Mi duda reside en que después se comenta que Linux, al tener representación de datos de 32 bits, puede tener problemas en el 2038. La cuestión que se me surge es: ¿a qué se debe esto?. Tiene que ver con la representación interna de las fechas o a la representación de datos. En caso de que se deba a esto último, otros S.O. (Windows 95, 98, Os/2 Warp, creo que también tienen representación de 32 bits), ¿tendrían los mismos problemas?.

Respuesta

El sistema operativo Linux y sus librerías disponen de un problema, sólo llegan a representar la fecha 19/01/2038. Esto se debe a que Linux guarda los datos y el tiempo como el número de segundos que han pasado desde las 0000 horas, Enero 1 de 1970. Al ser un sistema de 32 bits, el número máximo de segundos representables en un integro es de 2.147.483.647, y por ello, en el año 2038 llegará a su máximo.

Quedan 40 años hasta que el problema suceda y por entonces ya hará mucho tiempo que los sistemas dispondrán de 64 bits, por lo que estará resuelto por muchos años. Más concretamente la fecha representable será el año 292.271.025.015, mucho más allá de que cualquier usuario se preocupe por el retraso de la fecha en su sistema.

Pregunta

Soy lector de la revista, la cual sigo desde que apareció en la Argentina. El motivo de la presente, es si me pueden hechar una manita con un problema. La cuestión es que no sé cómo se instalan en forma manual los paquetes que están en los CD. ej. los entornos gráficos KAE y similares. Muchas gracias.

Respuesta

Bien. La instalación de cualquier gestor de ventanas para Linux es bastante sencilla, aunque reconocemos que al principio puede ser un poco frustrante y lioso. Lo primero que hay que hacer es compilar los fuentes del mismo, como nos dices que el paquete que deseas instalar es el KDE, pues suponemos que has obtenido el conjunto ya pre-compilado; en el caso de que esto no fuera así, acude al fichero **readme.txt** en el cual encontrarás indicaciones sobre cómo realizar la compilación. En la mayoría de los paquetes también se adjunta un *script* que realiza este proceso.

Una vez tengamos los ejecutables, sitúalos en un directorio aparte, puedes querer hacerlo en el /HOME-/GUI/KDE por ejemplo, aunque debes usar un directorio diferente para cada entorno obligatoriamente.

Segundo,crearemos un *script* personalizado para nuestro entorno, que llamado a su vez por **startx** en lugar del fichero por defecto situado en /etc/X11/xinit/xinitrc que nos cargaría el FVWM95. El fichero se localizará dentro del directorio /HOME y tendrá por nombre **.xinitrc** con las líneas:

```
#!/bin/sh
xterm &
exec [ruta completa del gestor de ventanas]
```

A partir de ahora, el *script* **startx** encontrará este fichero y hará uso de

él, arrancando las X-Windows con el nuevo entorno que le hallamos especificado en la ruta.

Pregunta

Hola, les escribo para ver si me pueden solucionar la siguiente duda. Hace poco que tengo una conexión a Internet y querría usarla desde mi sistema Linux (tengo instalada la distribución Eurielec Linux 98 que adjuntaba su revista). Seguí las instrucciones para conectarse del número 40 de su revista y, aunque desde consola establece la conexión, no consigo que Netscape reconozca la conexión. Lo he intentado con el kppp, pero me dice que la conexión acaba inesperadamente. El archivo de opciones y el script para la conexión son los que publicasteis (excepto la opción +ua, que no reconoce).

He intentado cargar pppd desde una terminal e iniciar mas tarde Netscape pero me da el siguiente error: "Netscape is unable to locate the server. The server does not have DNS entry". Me gustaría saber qué debo hacer para acceder a Internet desde Netscape en KDE. Gracias por todo y enhorabuena por su revista, sigan como hasta ahora.

Respuesta

Los mensajes de error que Netscape muestra en pantalla al establecer una conexión con el ISP (Servidor de Acceso a Internet) se deben a una mala configuración o creación de los scripts. Para resolver el problema, el primer paso a seguir es eliminar todos los scripts de conexión creados anteriormente. Deberían ser tres los eliminados, el script /etc/ppp/**options**, donde es guardada la información sobre la conexión, el /etc/ppp/**password** donde es guardado el login y la contraseña, y finalmente el /etc/ppp/**dial-isp**, donde es guardado el número del ISP. Los nombres de los scripts pueden variar así como el directorio donde se encuentran.

El segundo paso es crear el script /etc/ppp/**options** utilizando cualquier editor de textos, como podría ser vi. Seguidamente se debe copiar el siguiente texto tal y como está, sin eliminar la opción +ua, ya que debería ser reconocida. Una vez creado y guardado el script, se le debe aplicar la opción chmod 640 /etc/ppp/options.

```
#!/bin/sh
connect /etc/ppp/infovia
crtscts
modem
passive
+ua /etc/ppp/infoviappp
noipdefault
debug
defaultroute
asyncmap a0000
/dev/cuaX (Aquí se debe introducir el directorio
    donde el módem se encuentra)
38400 (Aquí se debe introducir la velocidad del
    módem)
```

El próximo paso es similar al anterior, también se deberá crear el script /etc/ppp/**infovia** y copiar el siguiente texto tal y como está. Es muy importante copiarlo exactamente igual. Si se dispone de acceso directo al ISP, donde pone 055, deberá introducir el número de teléfono del servidor de acceso a Internet. Una vez guardado se le debe aplicar la opción chmod 744 /etc/ppp/infovia.

```
#!/bin/sh
/usr/sbin/chat -v "" atdt055 CONNECT ""
```

El tercer script a crear es /etc/ppp/**infoviappp**, deberá introducir su login en donde pone su_login y su contraseña donde pone su_contraseña. Al igual que el primer script, deberá aplicar la opción chmod 640 /etc/ppp/infoviappp.

```
su_login
su_contraseña
```

Como último paso, deberá editar los scripts /etc/**rc.config** y /etc/-**resolv.conf** para asegurarse que la conexión se realiza con éxito. En el script rc.config, deberá buscar unas líneas parecidas a las siguientes y asegurarse de que ninguno de los IPADDR_ contiene una dirección IP, sólo deben estar las dos comillas (""). :

```
#
#IP Adresses
#
IPADRR_0=""
IPADRR_1=""
IPADRR_2=""
IPADRR_3=""
```

El siguiente paso es buscar la línea en donde se encuentra el siguiente texto:

```
SEARCHLIST="suservidor"
```

En esa línea deberá introducir el nombre de su servidor y su terminación. Por ejemplo, si su servidor es towercom.es, así debería quedar en rc.config:

```
SEARCHLIST="towercom.es"
```

La última línea a buscar es:

```
NAMESERVER=direcciónip
```

En lugar de direcciónip, deberá introducir la dirección IP que su servidor proporciona.

Como último paso, en el script /etc/**resolv.conf**, deberá editar la línea donde pone search o domain e introducir el nombre de su servidor y su terminación, como podría ser:

```
search towercom.es
```

También deberá introducir la dirección IP asignada por su servidor en la línea nameserver. Un ejemplo sería:

```
nameserver 194.179.1.100
```

Entornos de desarrollo

PROGRAMACIÓN

MICROSOFT DIRECTX 6.0 SDK

Este mes incluimos en exclusiva la herramienta que proporciona Microsoft a los desarrolladores para las implementaciones de APIs en programas multimedia y juegos. Para la ultimísima versión de DirectX, la 6.0, es necesario la desinstalación previa de versiones anteriores de este SDK.

SUN JAVA PC 1.0 BETA 4

Entorno para la creación de maquinas Network Computer. Se trata de uno de los entornos de desarrollo más novedosos creado por Sun. En fase beta, permite la creación de máquina virtuales que permiten la ejecución de cualquier programa en Java. Muy interesante para los seguidores de las últimas tendencias.

LENGUAJES

VISUAL BASIC

ACTIVEREPORTS 1.006

Herramienta profesional para generar informes en Visual Basic 5. Permite usar código Visual Basic, ActiveX y objetos OLE en los informes, lo que amplia las posibilidades de creación de casi cualquier tipo de documento. Incluye un visor que puede ser utilizado en Internet, soporta datos de tipo estático o dinámico (ASP).

DADVINAFENCE PARA VISUAL BASIC 5.0

Producto español que implementa un sistema de seguridad a sus desarrollos.

DadvinaFence es un sistema de seguridad activo fácil de integrar y totalmente configurable que puede ser implementado a los desarrollos realizados en Visual Basic 5.0.

JAVA

WEBSITE ABSTRACTION FREE JAVASCRIPTS

Colección de 10 JavaScripts de uso libre. Resulta muy interesante, dichos JavaScripts pueden ser utilizados libremente en cualquier Website. Incluyen desarrollos en Java de relojes, efectos de imagen, calendarios, links aleatorios, etc.

SUN JAVA DEVELOPMENT KIT 1.1.6

La última versión definitiva del entorno de desarrollo para Java de Sun

SUN JAVA DEVELOPMENT KIT 1.2 BETA 4

La última versión Beta del JDK de Sun. Además en el fichero comprimido x:\xx\xx podrás encontrar toda la documentación de esta versión. Los textos ocupan más de 14 Mb y están en formato HTML.

JAVA SCRIPT IT! 1.4

Colección de 20 efectos JavaScript que pueden ser añadidos a una página web.

Sin necesidad de tener conocimientos de programación, es posible añadir alertas, animaciones, presentaciones, mensajes de estado, botones, etc.

NETOBJECTS BEANBUILDER 1.0 PRE-RELEASE

Una excelente herramienta para la creación de JavaBeans.

HTML

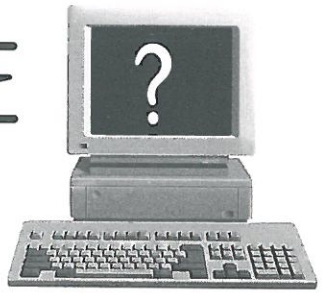
HOTDOG PRO WEBMASTER SUITE 5.0

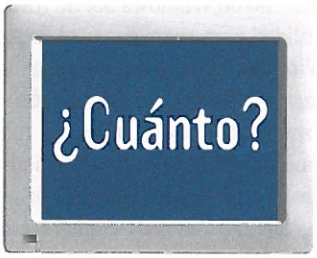
Además del editor, incluye un gestor de sitios web, puede analizar los enlaces perdidos, insertar enlaces, visualizar enlaces externos, imágenes y ficheros de texto, etc. HotDog puede editar scripts CGI y ficheros HTML, convertir ficheros de texto en HTML y trabajar con HTML dinámico. Además, puede crear plantillas y tags, descargar documentos o sitios completos, e incluye varios asistentes para canales, creación de índices de imágenes, etc. Se acompaña de herramientas para editar ficheros remotos, añadir plugins, crear botones, GIF´s animados, mapas de imágenes, animaciones y efectos de texto basados en Java y efectos JavaScript.

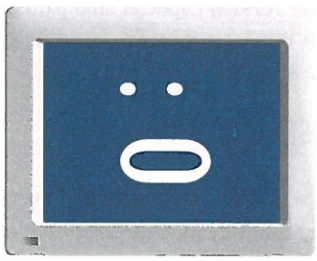
WEBSCRIPTER 1.5

Editor HTML con soporte especial para JavaScript que cuenta con menús y botones para crear tags HTML y un excelente soporte para crear scripts JavaScript. Incluye código fuente de ejemplo que puede ser modificado para

INTERNET ASEQUIBLE







crear eventos de cursor, funciones de escritura de documentos, botones, diálogos, nuevas ventanas, barras de estado e imágenes de enlace. Versión de evaluación válida para 20 usos.

XML EDITOR

Utilidad escrita en Java para la creación y edición de ficheros XML. El programa proporciona un interfaz gráfico que muestra la estructura del código HTML en forma de árbol. Permite arrastrar y soltar tanto para la creación como para la edición de tags XML.

MICROSOFT XML NOTEPAD 1.0 BETA

Editor al estilo del bloc de notas para la creación fácil de código XML. Se trata de un editor creado especialmente por Microsoft para el desarrollo de código XML para su navegador Internet Explorer 5.0. Incluye los tags más comunes predefinidos y permite una programación muy visual y rápida.

OTROS

ACTIVE PERL 502E

Entorno para la programación en Perl bajo Windows. Se trata de una herramienta freeware que facilita el desarrollo de aplicaciones 32 bits bajo Windows. Incluye Perl para Win32, Perl para ISAPI, PerlScript y el Perl Package Manager.

DADVINAFENCE PARA DELPHI 2

Producto español que permite implementar un sistema de seguridad a sus desarrollos. Logra entre otros, la asignación de permisos y privilegios sobre objetos existentes en las pantallas de las aplicaciones, autentificación de usuarios mediante el uso de identificación y contraseña, detección de intentos fallidos continuados de entrada al sistema avisando al instante a los administradores de seguridad de las aplicaciones, etc.

DADVINAFENCE PARA DELPHI 3

Permisos, contraseñas y privilegios en tus aplicaciones Delphi 3. Posee las mismas capacidades que el anterior.

LITESTEP 1.0 BETA

Shell para modificar la imagen del entorno de Windows 95. Se trata de una utilidad para reemplazar el Explorer de Windows 95 y especialmente indicada para usuarios que vienen del mundo Linux. Incorpora nuevos menús y modifica el estilo de las barras.Es necesario revisar la documentación antes de instalarlos. No debe copiarse bajo Windows 98 y requiere una instalación especial para Windows NT.

HERRAMIENTAS /UTILIDADES

DRIVERS DIRECTX 6.0

Última release de las rutinas multimedia y de juegos de Windows. Actualización de las APIs de Windows. Estas rutinas (DirectPlay, DirectDraw, DirectSound, DirectInput y Direct3D) mejoran ostensiblemente del sistema en programas multimedia y juegos. Para su instalación requiere aproximadamente 34 Mb. libres en el disco duro. Tras ésta, tan sólo ocupara 8 Mb. Es una versión especial sólo para usuarios de Windows 95 y 98. No debe instalarse sobre Windows NT.

GETRIGHT 3.20

Excelente utilidad para la automatización de tareas, especialmente indicado para la programación de descargas en Internet ya que es capaz de continuar la bajada de un fichero tras una desconexión. Permite la programación horaria de las descargas, su bajada automática, realiza la desconexión del equipo al finalizar la acción, etc. Imprescindible para cualquier usuario de Internet.

OPERA 3.21

Uno de los navegadores web más rápidos del mercado. Se basa en un interfaz con múltiples ventanas configurables, una completa lista de direcciones, etc. También ofrece prácticas opciones de correo y noticias (news).

CUENTAPASOS 3.11

Software para realizar el control del gasto telefónico. Interesante utilidad que permite realizar un seguimiento del gasto telefónico que genera nuestro módem al realizar llamadas a Internet. Esta ultimísima versión dispone de las modificaciones previstas en las nuevas tarifas implementadas por Telefónica este mes. Es imprescindible para los internautas más proclives. Preparado también para las tarifas de países sudamericanos

SISOFT SANDRA 98 V4.2.20

Programa de análisis y testeo que analiza el rendimiento del sistema. Analiza la configuración y el funcionamiento de los equipos que trabajan en Windows 95, 98 y NT. Proporciona información sobre el hardware, software y otros dispositivos: procesador, BIOS, Bus PCI, memoria Windows, memoria DOS, controladores, los puertos, tarjeta de sonido, dispositivos MCI, el rendimiento de la CPU, el vídeo, el CD-ROM, los ficheros de configuración del sistema, etc.

MCAFEE VIRUSSCAN 3.18E

Uno de los programas antivirus más potentes del mercado. Une a su velocidad de exploración, un avanzado sistema que combina distintas tecnologías

SÍ, ASEQUIBLE

NO ENCONTRARÁS UNA OFERTA MEJOR

¿Cómo?

Con un completo curso multimedia que te explicará desde en qué consiste el correo electrónico hasta qué es el FTP. Y, con conexión gratuita a Internet hasta el 31 de diciembre de 1.998.

¿Cuánto?

Por sólo 3.995 ptas. Iva incluido.

¿Dónde?

Podrás encontrarlo en librerías, quioscos, tiendas de informática, Vips, Crisol, Fnac, El Corte Inglés, Casa del libro o en el teléfono 91-6614211.

¿Cuándo?

Ya a la venta.

DOMINA
INTERNET 98
• APRENDA PASO A PASO A UTILIZAR INTERNET
• ELIJA EL HORARIO DE ESTUDIO SEGÚN SU TIEMPO DISPONIBLE
• ESTUDIE SIN ESFUERZO
• SERVICIO TÉCNICO DE ESTE PRODUCTO POR TELÉFONO, FAX Y CORREO ELECTRÓNICO
Incluye conexión a Internet
(finaliza el 31-12-98)
DOMINA
LA INFORMÁTICA

ABETO editorial

C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Tel.: (91) 661 42 11* Fax: (91) 661 43 86
http://www.abetoed.es

También versión sin conexión por 2.995 ptas.

de análisis, pudiendo así detectar y eliminar una gran variedad de virus. Detecta virus polimórficos y de macro. Incluye un modo "ScreenScan" que trabaja mientras el ordenador no está siendo utilizado.

WINZIP 7.0

La herramienta estándar para compresión y descompresión de ficheros. Se encarga de facilitar la compresión y descompresión de ficheros ZIP. Además, soporta numerosos formatos: TAR, GZ, ARJ, LZH, ARC, etc.

REDES

LOCALES

CABLEGUI! 1.01

Guía muy interesante que muestra cómo deben crearse cables informáticos. Se trata de una utilidad gráfica que permite conocer los trucos para la creación de cables informáticos. Muy útil para especialistas en redes e instaladores de éstas.

APACHE WEB SERVER 1.3.1

Uno de los programas servidor más utilizado en la implantación de webs. Es un estándar dentro de los programas de software servidor web. Incluye capacidades muy interesantes que posibilitan una sencilla instalación y un más fácil mantenimiento de las páginas web.

ORACLE PROXY SERVER 1.0

Software servidor proxy. Última versión del proxy de Oracle especialmente diseñado para Windows NT 4.0.

WEBSUITE SMARTDESK VIRTUAL MACHINE 3.5

Software servidor para la gestión de sitios web

DISTRIBUIDAS

WINGATE 2.0 PARA WINDOWS NT

Software proxy para la compartición de módems en una red. Última versión de uno de los programas para compartición de módems más conocido.

OTROS

MICROSOFT SITE SERVER EXPRESS 3.0

Software servidor para la gestión de sitios web. (www.microsoft.com).Pentium 200, 32 Mb.

PC-DUO 3.72

Programa de control remoto que permite acceder y controlar ordenadores a través de TCP/IP, NetBIOS o una red IPX. La pantalla del equipo aparece en una ventana de nuestro equipo, lo que proporciona un seguimiento total.

DOCUMENTACIÓN /TUTORIALES

C++ TUTORIAL

Curso de programación orientada a objetos con C++. Consiste en 12 lecciones en las que se puede aprender todo sobre punteros y funciones. Cada tema está presentado muy a fondo siguiendo test y preguntas para reforzar el sistema de aprendizaje. Incluye además muchos ejemplos de código fuente que pueden ser compilados al tiempo que se sigue el curso. El programa se compone de cuatro ficheros. El primero (cpphtm.zip) contiene las 12 primeras lecciones en formato HTML. El segundo (cppsrc.zip) los códigos fuentes de los programas. El tercero (cppans.zip) las respuestas a los ejercicios propuestos en el curso y el cuarto (flyaway.exe) es un juego DOS desarrollado en C++ a modo de ejemplo.

GUÍA DE APRENDIZAJE DE JAVA PC 1.0

Manual de referencia de la versión previa de desarrollo de Java PC. Se trata de una guía de referencia en formato HTML de la nueva versión del entorno de desarrollo para la creación de Network Computers de Sun. En este mismo CD, en el apartado Entornos, podrá encontrar la versión de evaluación de este producto, aún en su beta 4.

LEARN VISUAL BASIC 5 V1.0

Tutorial para abordar el aprendizaje personalizado de Visual Basic 5. Abarca desde conceptos de programación, terminología, herramientas, fundamentos de diseño, implementación y distribución de programas, etc. Se recomienda tener Visual Basic 5 Professional.

FUENTES DE LOS ARTÍCULOS

Programación en Java usando threads (II).

Todos los listados incluidos en el artículo.

Programación de Sockets (y V).

Las pruebas con CGI a la hora de la creación de un servidor web.

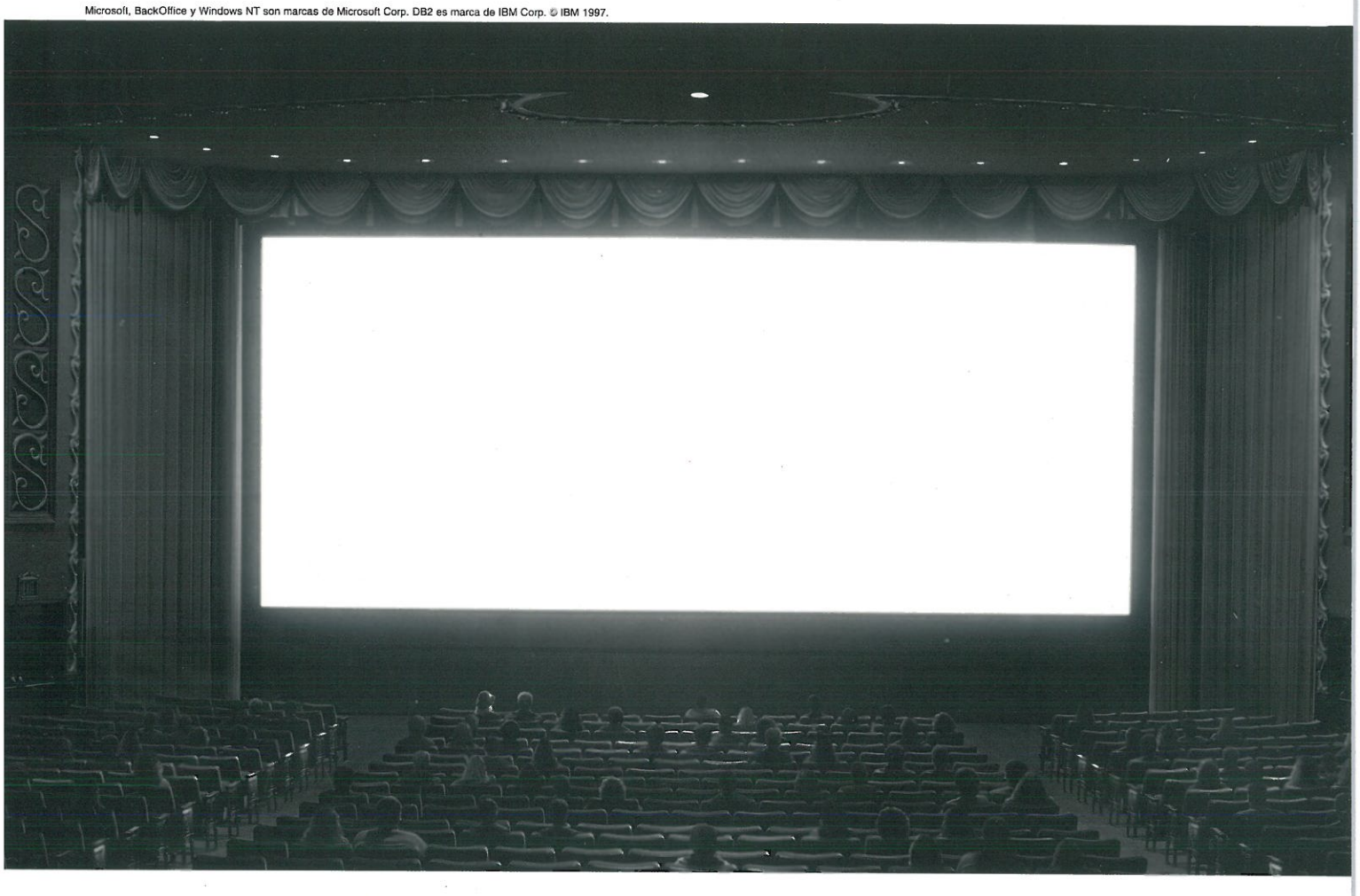
Programación de la SVGA (III)

Estos son los listados. Los programas los encontrarás en el directorio \program\svga3.

Diseñado para
Microsoft™ BackOffice™

DB2 para Windows NT es la única
Base de Datos Universal que puede llevar este distintivo.

Microsoft, BackOffice y Windows NT son marcas de Microsoft Corp. DB2 es marca de IBM Corp. © IBM 1997.



Sin DB2, su base de datos se está perdiendo lo mejor.

DB2 Universal Database le permite trabajar con imágenes y otros objetos como audio y vídeo, además de hacerlo con los datos tradicionales, incorporando estas nuevas posibilidades al ritmo que usted quiera. DB2 es el motor con el que podrá conseguir que la información cobre vida en Windows NT de forma excepcional: desde documentos escaneados o ficheros multimedia, hasta formación interactiva en Internet. Visítenos en www.software.ibm.com/info/db2nt y comprenderá lo mucho que se está perdiendo.

IBM

Soluciones para nuestro pequeño mundo